UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES
CAMPUS PATOS DE MINAS

ELTON SOARES SILVA

DESENVOLVIMENTO DE UM SISTEMA MÓVEL DE RECONHECIMENTO DE PLACAS DE VEÍCULOS BASEADO EM REDES NEURAIS ARTIFICIAIS

Patos de Minas - MG
2024

ELTON SOARES SILVA

DESENVOLVIMENTO DE UM SISTEMA MÓVEL DE RECONHECIMENTO DE PLACAS DE VEÍCULOS BASEADO EM REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso apresentado à banca examinadora como requisito de avaliação da disciplina de TCC2 da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, Campus Patos de Minas.

Orientador: Prof. Dr. Jeovane Vicente de Sousa.

Patos de Minas - MG

2024

ELTON SOARES SILVA

DESENVOLVIMENTO DE UM SISTEMA MÓVEL DE RECONHECIMENTO DE PLACAS DE VEÍCULOS BASEADO EM REDES NEURAIS ARTIFICIAIS

Trabalho de Conclusão de Curso apresentado à banca examinadora como requisito parcial de avaliação da disciplina de TCC2 da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, Campus Patos de Minas.

Orientador: Prof. Dr. Jeovane Vicente de Sousa.

Patos de Minas, 05 de abril de 2024

Banca Examinadora

_____

Prof. Dr. Jeovane Vicente de Sousa – FEELT/UFU (Orientador)

_____

Prof. Dr. Júlio Cézar Coelho – FEELT/UFU (Membro 1)

_____

Profª. Drª. Eliana Pantaleão – FACOM/UFU (Membro 2)

Dedico este trabalho aos meus pais,
Eleide e Ecival, que foram meus alicerces
para esta conquista.

AGRADECIMENTOS

Primeiramente, agradeço ao corpo docente, ao corpo técnico-administrativo e toda a comunidade acadêmica UFU – Patos de Minas por terem feito da UFU, um ambiente acolhedor e favorável ao desenvolvimento de cada aluno.

Ao meu orientador, Prof. Dr. Jeovane, por me nortear sobre como desenvolver um trabalho digno de conclusão de curso.

Aos demais membros da Banca Examinadora, por aceitarem o convite e participarem deste momento.

À Algar Telecom pela oportunidade de estágio, que me permitiu aprimorar minhas habilidades profissionais.

Aos meus amigos e colegas, que tornaram essa experiência mais enriquecedora, divertida e memorável.

Por fim, aos meus pais, pelo incentivo e apoio em trilhar este caminho.

RESUMO

Nos últimos anos, soluções baseadas em visão computacional têm emergido graças à evolução da capacidade de processamento dos computadores e das técnicas de visão computacional utilizadas, o que colaborou para implementá-las até mesmo em dispositivos diminutos, como os *smartphones*. Dentre as possíveis aplicações dentro deste campo, há o reconhecimento automático de placas veiculares, utilizado em estacionamentos, pedágios inteligentes, bem como em outras situações, em que um dispositivo recebe ou captura imagens de automóveis e reconhece a placa veicular de cada um, obtendo seus caracteres constituintes. Nesse sentido, este trabalho teve como objetivo desenvolver um sistema de reconhecimento automático de placas veiculares para dispositivos móveis capaz de integrar diferentes aplicações. Para isso, foram utilizados algoritmos e tecnologias de visão computacional, como o YOLO e Redes Neurais Convolucionais, juntamente com bases de imagens de veículos públicas, como o conjunto UFPR-ALPR, para treinar modelos capazes de detectar regiões contendo placas veiculares, segmentar os caracteres presentes nas placas identificadas e reconhecer cada um deles. Visando demonstrar um caso de uso, a aplicação utilizou as placas veiculares identificadas para consultar a situação de cada veículo, identificando se eles se encontravam em situação de furto ou roubo. Por fim, após avaliar a performance do sistema, foi obtido 70,70% de taxa de acerto na identificação de placas veiculares de imagens do conjunto de testes. Ou seja, a cada dez imagens de veículos, sete tiveram sua placa veicular identificada corretamente. Para isso, a aplicação gastou, em média, três segundos para processar cada imagem.

Palavras-Chave: visão computacional; reconhecimento automático de placas veiculares; Android; reconhecimento óptico de caracteres; redes neurais convolucionais; YOLO.

ABSTRACT

In the last years, solutions based on computer vision have emerged thanks to the evolution of the processing capacity of computers and the techniques used, which have helped to implement these solutions even in tiny devices such as smartphones. Among the possible applications inside this field, there is the automatic recognition of vehicle license plates, used in parking lots and smart tolls, as well as in other situations, where a device receives or captures images of cars and recognizes the license plate of each one, obtaining its constituent characters. In this sense, this work aimed to develop an automatic license plate recognition system for mobile devices capable of integrating different applications. For this, computer vision algorithms and technologies, such as YOLO and Convolutional Neural Networks, were used together with public vehicle image datasets, such as the UFPR-ALPR dataset, to train models capable of detecting regions containing license plates, segment the characters inside the identified plates and recognize each one of those characters. In order to demonstrate a use case, the application used the identified vehicle plates to consult the status of each one, identifying if the vehicles referring to them are in a situation of theft or robbery. In the end, after evaluating the system's performance, it was obtained 70,70% of success in identifying correctly the license plates from tests images set. That is, for every ten images containing a vehicle, seven license plates was correctly identified. To do this, the application spent, on average, three seconds to process each image.

Keywords: computer vision; automatic license plate recognition; Android; optical character recognition; convolutional neural networks; YOLO.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2-D	Duas Dimensões
3-D	Três Dimensões
ALPR	*Automatic License Plate Recognition*
AP	*Average Precision*
API	*Application Programming Interface*
AWS	Amazon Web Services
CNN	*Convolutional Neural Network*
FN	*False Negative*
FP	*False Positive*
GCP	Google Cloud Platform
GPU	*Graphics Processing Units*
HSV	*Hue*, *Saturation*, *Value*
HTTP	*Hypertext Transfer Protocol*
IoU	Interseção sobre a União
JSON	*JavaScript Object Notation*
mAP	*Mean Average Precision*
OCR	*Optical Character Recognition*
PR	Precisão-*Recall*
RGB	*Red*, *Green, Blue*
SINESP	Sistema Nacional de Informações de Segurança Pública
SVM	*Support Vector Machine*
TN	*True Negative*
TP	*True Positive*
UFPR	Universidade Federal do Paraná
YOLO	*You Only Look Once*

LISTA DE SÍMBOLOS

w_n	Enésimo Peso Sináptico
x_n	Enésimo Estímulo de Entrada
θ	Limiar de Ativação
u	Potencial de Ativação
g	Função de Ativação
y	Saída do Neurônio
S	Divisão do *grid*
B	Número de Caixas por Célula
C	Número de Classes Reconhecíveis pela Rede
N	Tamanho do *Kernel* de binarização com limiar adaptativo
n	Número de *Thresholds*
n_c	Número de Classes
A	Número de Caixas Âncora
C	Número de Classes

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Neste capítulo introdutório, serão apresentados o contexto e tema deste projeto, bem como as motivações, objetivos e hipóteses para este trabalho.

1.1 INTRODUÇÃO

Algoritmos capazes de construir sistemas com certo grau de inteligência vêm sendo utilizados para resolver problemas de diversas áreas, inclusive em diferentes campos da Engenharia, tornando-se uma ferramenta universal que estudiosos de diferentes esferas do saber precisam estar atentos.

Esses sistemas são construídos a partir dos conhecimentos das diferentes subáreas que existem dentro da área de Inteligência Artificial, que, por sua vez, costuma ser definida como a ciência e engenharia que existe por trás do desenvolvimento de sistemas inteligentes (1). Nesse contexto, inteligência costuma ser definida de diferentes formas pelos pesquisadores, com alguns colocando este termo em função da fidelidade da reprodução da inteligência dos seres humanos, enquanto outros associam à racionalidade, que é uma definição associada ao conceito de realizar da maneira “correta”, as tarefas as quais o modelo de Inteligência Artificial tenha sido projetado para executar (2).

Dentre as diferentes subáreas, conceitos e tópicos da Inteligência Artificial, alguns que aparecem com frequência são (1):

a) Procura;
b) Inferência;
c) Aprendizado por meio da Experiência;
d) Algoritmos Genéticos;
e) Reconhecimento de Padrões;

A primeira subárea se refere aos programas em que se busca o caminho mais eficiente dentre um grande número de formas possíveis para se tratar um problema.

Inferência, por sua vez, diz respeito aos códigos construídos para se obter conclusões a partir de outras conclusões. Algoritmos baseados em Aprendizado por meio da Experiência tratam de programas baseados em redes neurais e conexionismo, utilizando regras de aprendizado para aprender com as informações apresentadas ao programa. Algoritmos genéticos, como o próprio nome indica, são códigos em que a resposta para um problema é obtida por meio da seleção da solução mais adequada dentre várias outras que foram geradas com códigos aleatórios. Reconhecimento de Padrões, por seu turno, é baseado em códigos que comparam dados, como imagens, sons, dentre outros tipos, com outros que já tenham sido apresentados anteriormente ao programa, com o objetivo de identificar discrepâncias e semelhanças entre eles (1).

Dentro da seara de Reconhecimento de Padrões, existem as tecnologias de visão computacional (1), que tratam de ensinar computadores a interpretar imagens de maneira similar à visão humana (3). A visão computacional vem sendo bastante utilizada para a solução de problemas devido a sua capacidade de habilitar sistemas a analisar imagens mais rapidamente do que seres humanos, e perceber detalhes imperceptíveis aos olhos humanos. Dentre suas vantagens estão o desenvolvimento de veículos autônomos, a identificação de falhas em linhas de produção, o reconhecimento e tradução de textos, a identificação de máquinas necessitando de manutenção, dentre outras aplicações. Dessa forma, considerando as possibilidades de uso e vantagens que a visão computacional é capaz de promover, é esperado que o mercado global envolvendo seu uso tenha atingido o valor de US$ 48,6 bilhões em 2022 (3).

Uma importante aplicação de visão computacional é o reconhecimento automático de placas veiculares ou *Automatic License Plate Recognition* (ALPR), que pode ser usado em sistemas de assistência de estacionamento, pedágios automatizados, sistemas de entrega e logística em portos, dentre outros cenários em que é necessário realizar o reconhecimento de placas veiculares (4). A Figura 1.1 demonstra um sistema destes, sendo possível ver que a aplicação conseguiu identificar as porções da imagem contendo o veículo e a placa veicular, bem como realizar a identificação completa dos seus caracteres constituintes.

Figura 1.1 – Exemplo de um Sistema ALPR em Funcionamento

Fonte: Adaptado de: (4).

Sistemas desse tipo podem envolver etapas como detecção de veículos, detecção de placa veicular e reconhecimento de caracteres, como a Figura 1.2 ilustra, sendo a primeira etapa não obrigatória, visto que alguns sistemas conseguem identificar a região contendo a placa sem precisar segmentar o veículo primeiro.

Figura 1.2 – Exemplo de Etapas de Desenvolvimento de um Sistema ALPR

Fonte: Adaptado de: (4).

Nesse sentido, o desenvolvimento desse tipo de projeto costuma envolver modelos de *deep learning* para detecção de objetos, e reconhecimento óptico de caracteres ou *optical character recognition* (OCR) (4), que são, tipicamente, onerosos

em termos computacionais, o que torna a implementação desses tipos de sistemas em dispositivos de baixa capacidade computacional, como *smartphones*, um desafio em que o projetista deverá procurar pelo equilíbrio entre acurácia dos resultados obtidos pelo sistema, tempo de execução dos algoritmos, e recursos computacionais disponíveis, buscando pelas melhores tecnologias que auxiliem nessa questão.

1.2 TEMA DO PROJETO

O tema deste trabalho é o uso de ferramentas de visão computacional e reconhecimento óptico de caracteres em dispositivos móveis, para realizar a localização de placas veiculares e o reconhecimento de seus caracteres, isto é, a construção de um sistema ALPR para *smartphones*.

1.3 PROBLEMATIZAÇÃO

Como destacado anteriormente, ferramentas de visão computacional possuem um vasto campo de aplicações, podendo sanar problemas em diferentes áreas. Na seara de Segurança Pública, por exemplo, a cidade de Praia Grande, em São Paulo, implantou um sistema de reconhecimento facial para identificar pessoas procuradas pela Justiça, colhendo frutos como redução da taxa de criminalidade nas áreas monitoradas (5). Por outro lado, na área da saúde, esta tecnologia tem habilitado a construção de ferramentas, como o DermAssist, aplicação desenvolvida pelo Google para detecção de câncer de pele utilizando imagens coletadas por *Smartphones* (6).

No entanto, o potencial desta tecnologia para a tratativa de problemas segue sendo subutilizado. Por exemplo, um problema comum no Brasil é o furto e roubo de veículos. Somente entre 2015 e 2019, segundo dados do Sistema Nacional de Informações de Segurança Pública (SINESP), houve mais de dois milhões de ocorrências neste sentido (7). Portanto, considerando que veículos furtados ou roubados no Brasil são cadastrados em uma base nacional de dados que discrimina automóveis furtados ou roubados de veículos regulares (8), é possível aliar o uso de visão computacional com estas informações para desenvolver medidas para identificar e apreender veículos irregulares circulando em vias públicas. Assim, uma ação que pode ser realizada é a utilização de câmeras de sistemas de vigilância

pública, como as câmeras pertencentes ao Projeto Olho Vivo, implantado em Patos de Minas (9), para capturar imagens de veículos e repassá-las para sistemas ALPR realizarem a identificação e verificação da situação de cada automóvel encontrado, gerando alertas, quando necessário, para as autoridades locais atuarem da maneira mais adequada. Somando-se a isso, também poderiam ser feitas parcerias com entes privados para utilizar câmeras de comércios e residências, além de poder ser desenvolvida uma aplicação para dispositivos móveis, em que o próprio cidadão poderia apontar a câmera do *smartphone* para um automóvel e descobrir sua situação.

Portanto, há diversos cenários em que o uso de visão computacional pode ser empregado para auxiliar na resolução de problemas. Dessa forma, para estimular o desenvolvimento destas soluções, é necessário difundir as técnicas e o conhecimento utilizado neste campo, habilitando as pessoas a empregar visão computacional para resolver problemas.

1.4 JUSTIFICATIVAS E HIPÓTESES

Dessa forma, este trabalho se justifica por atuar como prova de conceito para demonstrar como técnicas e ferramentas de visão computacional podem ser utilizadas como soluções em diferentes problemas, utilizando como base a questão do roubo e furto de veículos no Brasil. Assim, a hipótese levantada é a capacidade de realizar a identificação completa dos caracteres de placas veiculares presentes em fotos contendo automóveis, de maneira satisfatória, isto é, com boa acurácia e sem gasto de elevada quantidade de tempo para identificar uma única placa, utilizando *smartphones* para esta tarefa.

Considerando particularidades das placas veiculares brasileiras, como a disposição dos caracteres constituintes em uma única linha ou em duas, de acordo com o tipo do veículo, foi definido que o sistema construído terá, como foco, o reconhecimento correto dos caracteres para veículos com placas contendo apenas uma linha de caracteres, como é o caso de carros, ônibus, caminhões, dentre outros tipos de veículos.

1.5 OBJETIVOS

Nesta seção serão apresentados os objetivos gerais e específicos deste trabalho.

1.5.1 Objetivos Gerais

O objetivo geral deste trabalho é desenvolver uma aplicação para *smartphones* Android, capaz de reconhecer placas veiculares e seus caracteres em uma imagem que contenha pelo menos um automóvel com a placa veicular completamente visível, de forma a demonstrar como pode ser feito o uso de visão computacional para resolver problemas.

1.5.2 Objetivos Específicos

Os objetivos específicos, complementares e necessários para cumprir os objetivos gerais, são:

a) Realizar a captura de imagens de veículos, utilizando dispositivos móveis;
b) Localizar as placas veiculares presentes nas imagens obtidas, utilizando técnicas de visão computacional;
c) Reconhecer corretamente os caracteres das placas veiculares obtidas;
d) Realizar os objetivos anteriores de maneira aceitável, em um *smartphone* de capacidade computacional comparável a modelos de entrada e intermediários, para placas veiculares contendo uma única linha de caracteres.

1.6 CONSIDERAÇÕES FINAIS

Dessa forma, motivado pelo intuito de demonstrar a aplicabilidade da visão computacional para a resolução de problemas, ao final deste trabalho, pretende-se obter uma ferramenta para dispositivos móveis, capaz de receber ou capturar imagens de veículos, identificar a região contendo a placa veicular, segmentar e identificar

todos os seus caracteres de maneira satisfatória para placas contendo seus caracteres em uma única linha. Uma vez que o escopo do trabalho foi definido, bem como seu contexto e motivações, no próximo capítulo será apresentado o referencial teórico utilizado para este trabalho.

CAPÍTULO 2

REFERENCIAL TEÓRICO

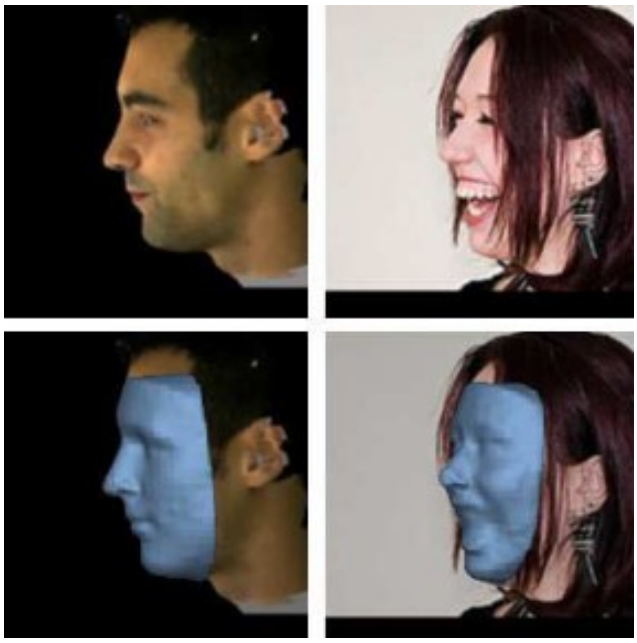
A seguir, será apresentada a base teórica pertinente a este trabalho. Para isso, projetos desenvolvidos na mesma linha irão servir de referência, bem como serão discutidos conceitos que irão nortear este trabalho.

2.1 VISÃO COMPUTACIONAL

Como mencionado anteriormente, visão computacional diz respeito a implementar nas máquinas algo similar ao sentido da visão, presente nos seres humanos. Soluções utilizando essa tecnologia, tipicamente, enfrentam problemas de reconstrução ou de reconhecimento. O primeiro se refere à construção de modelos utilizando uma imagem ou um conjunto de imagens. O segundo, por sua vez, está associado a fazer distinção entre objetos em uma imagem baseado nas informações visuais ou em outras informações (2).

Um exemplo do primeiro caso é o problema de reconstrução de faces em três dimensões (3-D) utilizando imagens em duas dimensões (2-D), como em (10), em que os autores utilizam uma Rede Neural Convolucional ou *Convolutional Neural Network* (CNN) treinada com imagens de faces em 2-D, e modelos faciais em 3-D, para receber uma única imagem em 2-D e reconstruí-la em um modelo facial em 3-D. Nesse caso, a CNN é utilizada para realizar uma regressão direta da representação volumétrica de uma face em três dimensões utilizando uma face em apenas duas dimensões, modelando, até mesmo, partes não visíveis na imagem em 2-D (10). A Figura 2.3 ilustra os resultados obtidos no trabalho mencionado, sendo visível na primeira linha, as imagens originais, e na segunda linha há uma sobreposição entre a face 3-D, obtida pelo modelo treinado pelos autores, e as imagens originais.

Figura 2.3 – Exemplo de Reconstrução de Face em 3-D utilizando Faces 2-D



Fonte: Adaptado de: (10).

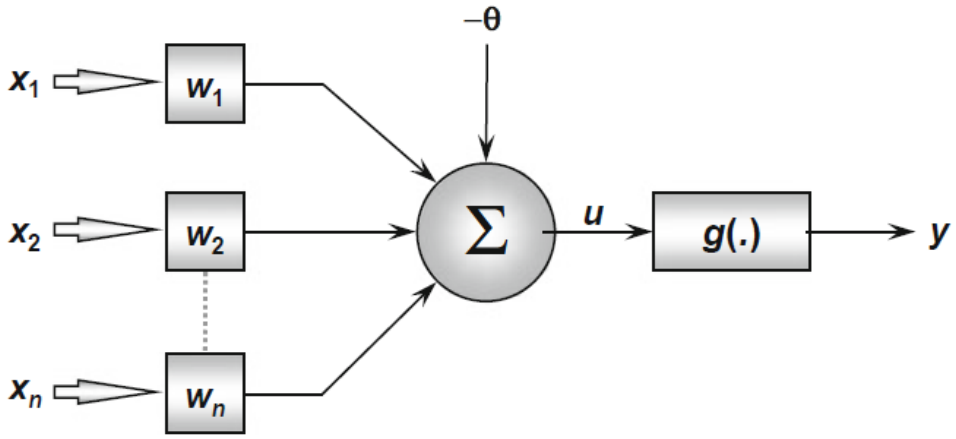
Um exemplo para o segundo caso é, justamente, o problema de localização de placa veicular, exemplificado na Figura 1.2, em que uma imagem de um veículo deve ser analisada em busca da região que contém a placa, para que essa região seja enviada para o tratamento de algum algoritmo de reconhecimento de caracteres. Em (11), é implementado um sistema de reconhecimento de placa veicular para dispositivos *mobile*, porém, utilizando um OCR externo para fazer o reconhecimento de caracteres, cabendo à aplicação desenvolvida pelos autores apenas realizar a extração e tratamento da região contendo a placa, e enviá-la para o OCR. Para isso, os autores utilizam a técnica *You Only Look Once* (YOLO), baseada também em CNN, por se tratar de uma tecnologia capaz de aliar alta acurácia de resultados com boa performance em dispositivos com limitados recursos computacionais (11).

2.2 REDES NEURAIS

Como visto nos dois trabalhos citados anteriormente, redes neurais têm sido de grande uso em aplicações de visão computacional. Elas se tratam de ferramentas desenvolvidas para se assemelhar ao sistema nervoso humano, capazes de adquirir e guardar conhecimento. As redes neurais são construídas por meio de unidades de processamento, chamadas de neurônios artificiais, interconectadas entre si (12).

Em termos matemáticos, um modelo de neurônio artificial, como o modelo proposto por McCulloch e Pitts em 1943 (13) e exemplificado na Figura 2.4, é representado por pesos sinápticos (w_1, w_2,..., w_n), que são multiplicados pelos estímulos de entrada (x_1, x_2,..., x_n) para, depois, os resultados serem somados pelo agregador linear dentro do neurônio, juntamente com o limiar de ativação ou *bias* (θ), gerando o potencial de ativação (u), que é recebido pela função de ativação (g), cuja saída é o sinal de saída do neurônio, (y), (12).

Figura 2.4 – Neurônio Artificial Proposto por McCulloch e Pitts



Fonte: (12).

Dessa forma, o funcionamento do neurônio artificial pode ser resumido pelas equações (2.1) e (2.2) vistas a seguir (12):

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \theta \tag{2.1}$$

$$y = g(u) \tag{2.2}$$

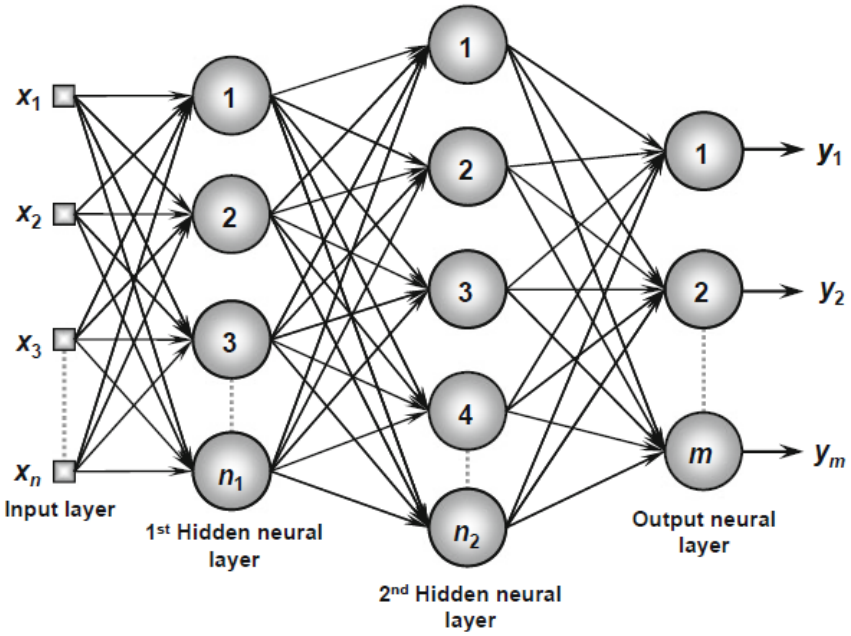
Cabe destacar que existem diversas funções de ativação diferentes, como as funções tangente hiperbólica, degrau ou Heaviside, degrau bipolar, rampa simétrica, dentre outras, sendo cada uma mais apropriada para ser utilizada diferentes tipos de problemas. Por exemplo, a seguir, na equação (2.3), tem-se a função de Heaviside, muito utilizada para problemas de reconhecimento de padrões, em que a saída da rede neural deve indicar a qual classe os sinais de entrada pertencem (12), como falso

ou verdadeiro, preto ou branco, enfermo ou saudável, dentre outros tipos de classes, não se restringindo a problemas com somente duas classes.

$$g(u) = \begin{cases} 1, & se\ u \geq 0 \\ 0, & se\ u < 0 \end{cases} \tag{2.3}$$

Uma rede neural típica, como a Rede Neural Multicamadas proposta por Frank Rosenblatt em 1958, também chamada de Perceptron multicamadas (14), é composta por diversos neurônios interconectados. Como pode ser visto na Figura 2.5, a rede tem a camada de entrada, com os sinais externos que irão excitar a rede neural, as camadas ocultas, compostas por neurônios que irão enviar seu sinal de saída para excitar os neurônios das camadas seguintes até chegar na camada de saída, que irá entregar a resposta da rede neural. Esta resposta pode ser a detecção ou não de alguma característica. Por exemplo, os sinais de entrada poderiam ser os níveis de intensidade dos *pixels* de uma imagem, enquanto a saída poderia ser a existência ou não de um objeto na imagem.

Figura 2.5 – Exemplo de Rede Neural Multicamadas

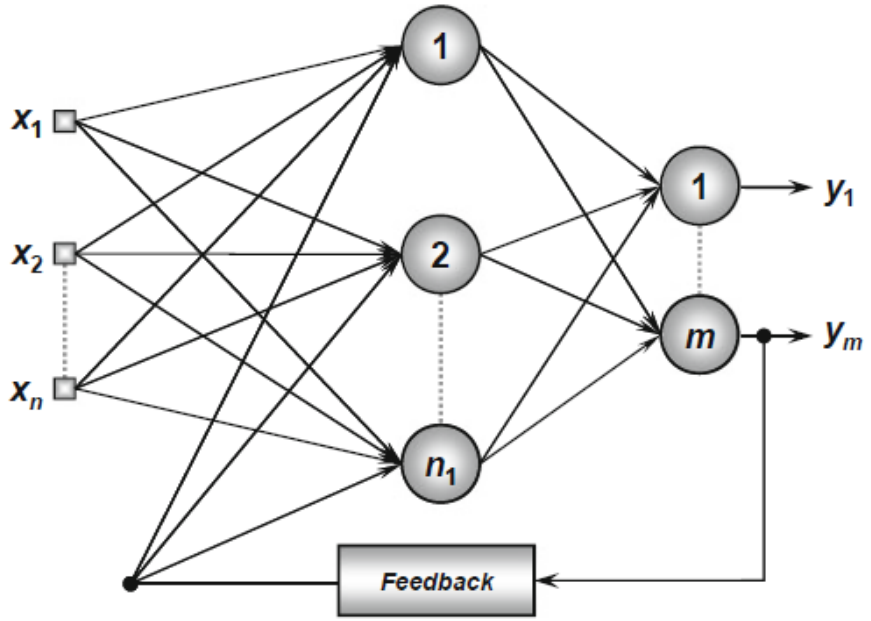


Fonte: (12).

Cabe salientar que existem diferentes arquiteturas de redes neurais, sendo a Rede Neural Multicamadas somente uma dentre várias. Por exemplo, há as redes

neurais recorrentes ou com arquitetura de realimentação, que se caracterizam por realimentar o sinal de saída de uma camada de neurônios em uma outra camada anterior, tornando estas redes mais adequadas para lidar com problemas envolvendo sistemas variantes no tempo (12). A rede da Figura 2.6 é um caso de rede neural com realimentação, em que os sinais de saída dos neurônios da última camada são realimentados nos neurônios da primeira camada escondida.

Figura 2.6 – Rede Neural Recorrente ou com Arquitetura de Realimentação



Fonte: (12).

2.2.1 Treinamento de Redes Neurais

Para uma rede ser aplicada na resolução de algum problema, antes ela precisa passar por um processo de treinamento para ajustar os valores dos pesos sinápticos e dos limiares de ativação. Nesse sentido, há diversos tipos de treinamento, como o Supervisionado, Não-supervisionado, por Reforço, *Offline* e *Online* (12).

No treinamento Supervisionado, os dados de entrada e os dados correspondentes de saída são apresentados, então, o algoritmo busca ajustar pesos e limiares para aproximar a saída da rede à saída desejada. No treinamento Não-Supervisionado, somente os dados de entrada são apresentados, e a rede busca encontrar grupos de dados similares entre si, realizando um processo comumente

chamado de “clusterização”. O Treinamento por Reforço é similar ao treinamento supervisionado, com a diferença que a rede não tem um pacote de dados de entrada e dados de saída correspondente, no máximo, a rede tem o conhecimento se a saída para determinada entrada é satisfatória ou não, exigindo ajustes nos pesos e limiares, que podem aproximar ou afastar a saída da rede do resultado desejado. Dessa forma, é comum dizer que a rede é treinada por tentativa e erro. O treinamento *offline*, por sua vez, é aquele em que os ajustes nos pesos e limiares são feitos somente após todos os dados de treinamento terem sido apresentados à rede, enquanto o *online* é baseado em realizar os ajustes a cada amostra apresentada (12).

Um algoritmo de treinamento bastante utilizado é o *backpropagation*. O Procedimento para aplicar esta técnica foi, inicialmente, apresentado em 1986 (15), no entanto, hoje, esta técnica possui diversas versões otimizadas. Este algoritmo possui este nome pois consiste em inserir os dados de entrada e obter a saída da rede para, depois, compará-la à saída desejada, obtendo a diferença entre ambas e propagando-a de volta, pela rede neural, desde a camada de saída até a primeira camada de neurônios, recalculando todos os valores de pesos sinápticos e limiares de ativação, inclusive dos neurônios pertencentes às camadas ocultas, de modo a minimizar a diferença entre a saída desejada e a saída obtida da próxima vez que os mesmos dados passarem pela rede (12). Dessa forma, o *backpropagation* pode ser definido como uma regra de propósito geral relativamente simples, capaz de construir representações internas do problema que a rede neural deverá ser capaz de resolver (15).

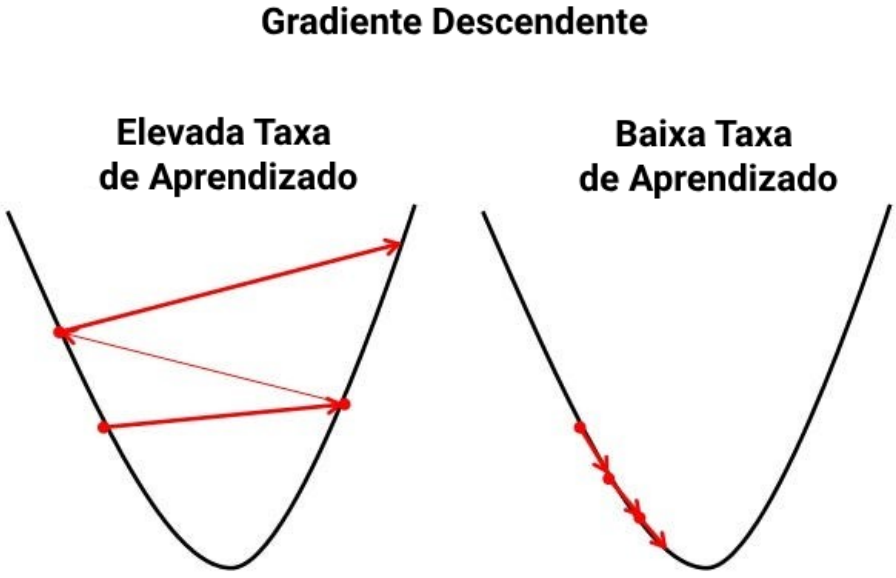
Para treinar uma rede neural, deve-se definir os valores dos hiperparâmetros, que são variáveis que definem como a rede será treinada e como ela foi construída. Referente ao treinamento da rede, um dos hiperparâmetros é o “*batch size*” ou tamanho do lote, em tradução livre, que define quantos elementos do conjunto de treinamento serão utilizados antes que os pesos e limiares da rede sejam atualizados. Por sua vez, o número de épocas representa quantas vezes o conjunto de dados de treinamento será completamente utilizado no treinamento da rede (16).

Também há a taxa de aprendizado ou “*learning rate*”, que indica o tamanho do passo utilizado para atualizar os pesos e limiares da rede. Valores elevados para este hiperparâmetro permitem a rede aprender mais rapidamente, porém, dificulta a convergência para um ponto em que o erro entre a saída esperada e a saída da rede é mínimo. Valores pequenos facilitam a convergência em torno de um mínimo, mas

diminuem a velocidade de aprendizado (16). Ou seja, utilizar baixos valores para a taxa de aprendizagem pode exigir mais épocas de treinamento para que a rede atinja um ponto em que o erro é mínimo.

A Figura 2.7 ilustra a diferença entre taxas baixas e elevadas de aprendizado. Os pontos vermelhos na curva esquerda representam o erro entre saída esperada e saída obtida. Como o valor do passo é elevado, a rede não consegue convergir em torno do mínimo local desta curva para minimizar o erro. A curva direita representa uma situação em que a taxa de aprendizado é baixa. Por conta disso, a rede consegue convergir para o mínimo local, mas precisa de várias épocas para isso. Uma forma de lidar com esta questão é utilizando algoritmos de treinamento com taxas de aprendizado adaptativas. Assim, o valor da taxa diminui ou aumenta de acordo com a evolução do gradiente de perdas (17), isto é, se a taxa de erro se aproxima do mínimo local, a taxa de aprendizado diminui, se o erro aumenta, a taxa de aprendizado também aumenta.

Figura 2.7 – Comparação entre Taxa de Aprendizado Elevada e Baixa



Fonte: Adaptado de: (16).

Por sua vez, o *momentum* é um hiperparâmetro cujo valor varia de 0 a 1 e atua ponderando o quanto os pesos e limiares da rede se alteraram entre uma iteração e outra (16). Se a mudança nos valores for grande, significa que o erro entre saída esperada e obtida ainda é grande. Portanto, o *momentum* incrementa ainda mais as

mudanças nos pesos e limiares. Porém, caso a diferença seja pequena, haverá pouca influência na atualização dos parâmetros da rede. Dessa forma, este parâmetro promove uma convergência mais eficiente em torno de um ponto de mínimo (16).

Referente aos hiperparâmetros de estrutura da rede, há o número de camadas ocultas e neurônios artificiais em cada camada. Aumentar estes números pode melhorar, até certo ponto, os resultados obtidos pela rede (16). Após determinado ponto, o acréscimo de neurônios e camadas pode causar o *overfitting*, que é quando o erro entre saída desejada e obtida para o conjunto de treinamento diminui, mas para o conjunto de testes, aumenta. Na situação de *overfitting*, a rede memoriza a saída correta para os sinais de entrada utilizados no treinamento, consequentemente, o erro observado para o conjunto de treinamento é pequeno. Porém, a capacidade de generalização da rede é prejudicada, de modo que, para sinais de entrada diferentes daqueles utilizados durante o treino, a rede produz elevada taxa de erro (12).

Diminuir o número de neurônios e camadas pode causar *underfitting* (16), que ocorre quando o modelo não é capaz de extrair e armazenar as características do processo para qual está sendo treinado (12), isto é, mesmo após várias épocas de treinamento, a taxa de erro não apresenta melhoras significativas.

Outro parâmetro que pode ser configurado no treinamento de um modelo é o *dropout*, que representa uma porcentagem de neurônios pertencentes a uma determinada camada, escolhidos de forma aleatória, para não serem utilizados durante uma época de treinamento. Esta estratégia visa conferir maior capacidade ao modelo para aprender representações independentes dos dados inseridos na rede neural, evitando o *overfitting* (16).

A inicialização dos valores dos pesos e limiares e as funções de ativação utilizadas também são hiperparâmetros. Tipicamente, os pesos e limiares são inicializados de forma aleatória, utilizando distribuição uniforme (16). Já a função de ativação, como comentado anteriormente, depende do tipo de problema tratado.

2.2.2 Redes Neurais Convolucionais (CNN)

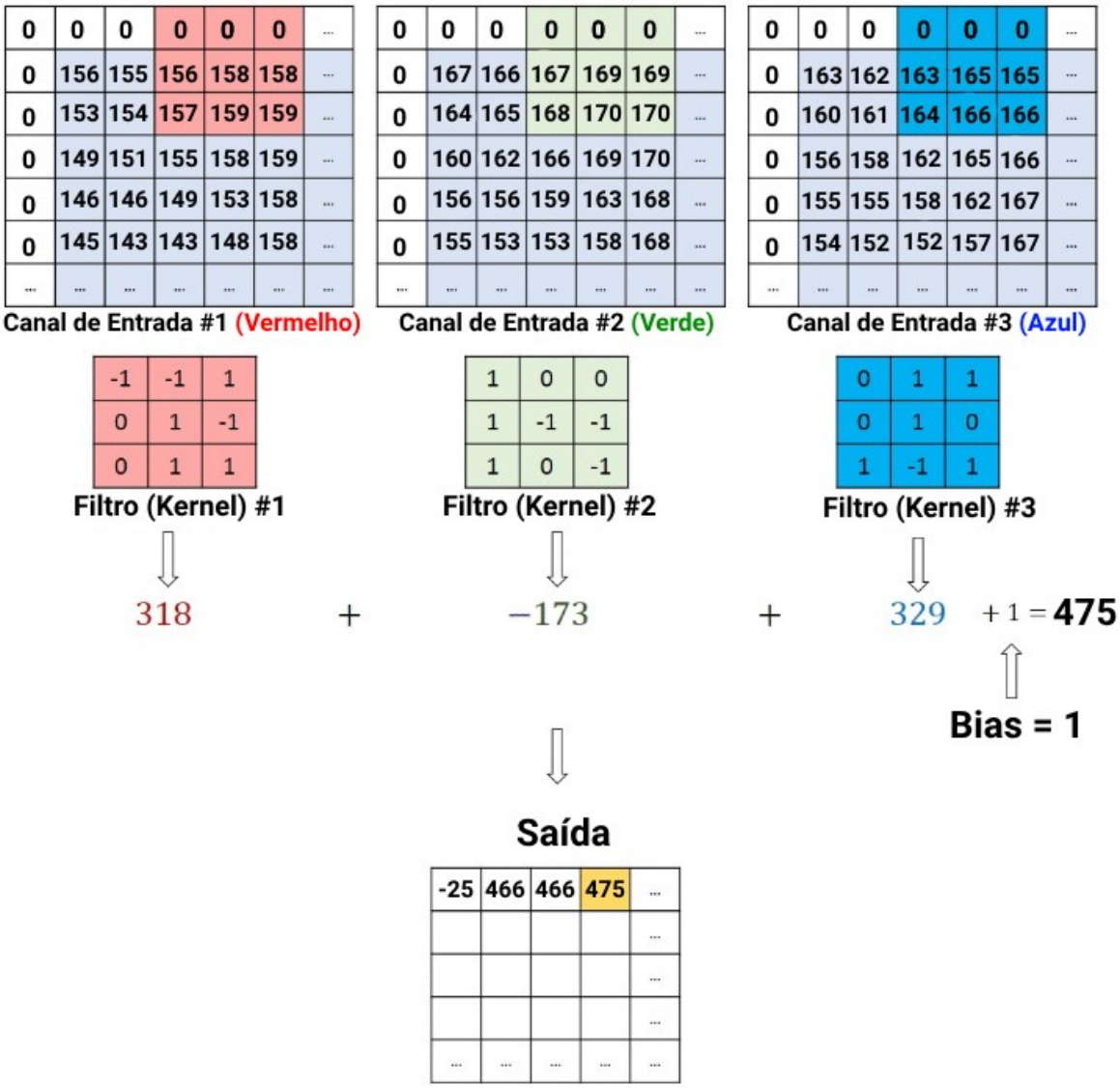
As Redes Neurais Convolucionais (CNN) são arquiteturas bastante utilizadas em problemas envolvendo imagens, visto que elas possuem desempenho tipicamente superior a outros tipos de Redes Neurais Artificiais (2) a um custo computacional menor, isto é, processando imagens mais rapidamente do que outros tipos de redes (11).

Para obter resultados melhores do que outros tipos de redes, a arquitetura da Rede Neural Convolucional, apresentada em 1998 (18), inovou ao introduzir camadas convolucionais e camadas de *pooling*, juntamente com as camadas totalmente conectadas entre si, como aquelas vistas no Perceptron multicamadas (19), para realizar tarefas de reconhecimento de padrões, como reconhecimento de caracteres escritos à mão (18).

Nas camadas de convolução, estruturas conhecidas como *kernels*, também chamadas de filtros, fazem a extração de características da imagem por meio da operação de convolução entre a matriz que representa o filtro, e as matrizes que representam diferentes regiões da imagem (19). A Figura 2.8 ilustra este processo, em que as matrizes “Canal de Entrada #1 (Vermelho)”, “Canal de Entrada #2 (Verde)” e “Canal de Entrada #3 (Azul)” representam os três canais de cores de uma imagem no sistema RGB (vermelho, verde e azul). Após realizar a convolução entre diferentes regiões da imagem com o filtro ou *kernel*, é obtida a matriz de saída, contendo o resultado da filtragem da imagem realizada na camada convolucional.

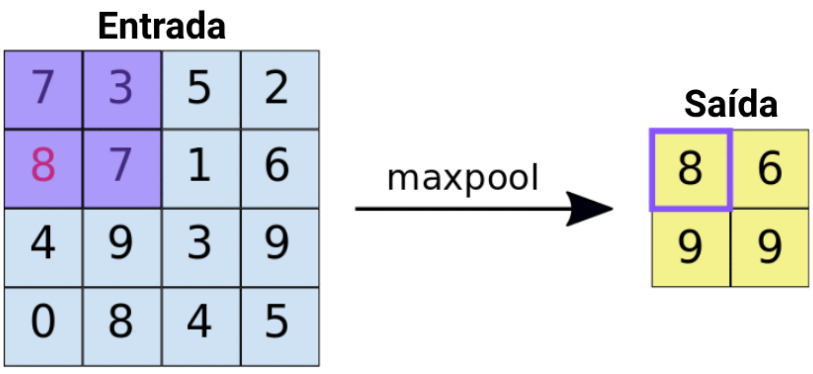
Por sua vez, nas camadas de *pooling* é realizado o *downsampling*, ou subamostragem, da imagem que passa por esta camada, de modo a reduzir seu tamanho (2). A Figura 2.9 ilustra este processo, com a matriz de Entrada e de Saída indicando a imagem antes e depois de passar pela camada de *pooling*. Para executar a redução da imagem, uma matriz 2 por 2 foi deslocada por cada região da matriz de entrada, calculando um valor para representar cada região. Para realizar este processo, diferentes metodologias de cálculo podem ser empregadas, sendo tipicamente escolhido o método *maxpooling*, que foi empregado na Figura 2.9, em que o *pixel* de maior valor dentro da matriz 2 por 2 é escolhido para representar a região, enquanto os demais são descartados (19).

Figura 2.8 – Exemplo do Processo de Convolução para uma Imagem RGB



Fonte: Adaptado de (19).

Figura 2.9 – Exemplo de Funcionamento da Camada de *Pooling*



Fonte: Adaptado de (19).

Dessa forma, ao reduzir o tamanho da imagem, estas camadas contribuem para diminuir a quantidade de informações relativas ao sinal de entrada. Por consequência, a quantidade de cálculos necessários para a rede neural realizar o processamento de uma imagem inserida em sua entrada também se reduz (11), o que contribui para que esta arquitetura tenha melhor performance computacional para processar imagens, quando comparado a outros tipos de redes neurais artificiais.

2.2.3 Algoritmo *You Only Look Once* (YOLO)

É um algoritmo de código aberto (*open source*), apresentado em 2015 (20), que possui diferentes versões feitas pelos autores originais e por terceiros, como as versões YOLOv5, YOLOv4, dentre outras.

Como os autores do algoritmo original, YOLOv1, destacam em seu trabalho, esta tecnologia consegue não somente detectar se uma imagem contém um ou mais objetos, mas consegue detectar a localização de cada um (20). Para isso, uma única CNN é usada para encontrar simultaneamente as regiões contendo objetos, e suas categorias, processando toda a imagem uma única vez (20). Assim, é dito que o YOLO é um algoritmo de passada única (*single pass*), ou seja, que “olha” para a imagem uma única vez para obter os resultados, sendo que é esta característica que é utilizada para dar nome ao algoritmo (21).

Para detectar os objetos, a imagem é dividida em células, em que cada uma é responsável por detectar o objeto dentro dela. Nesse sentido, para cada célula, são calculadas as posições e dimensões de regiões, chamadas de caixas delimitadoras, que visam delimitar a região de um objeto. Essas regiões podem ter largura e altura maiores do que as dimensões da própria célula, visto que podem haver objetos maiores que uma única célula (20).

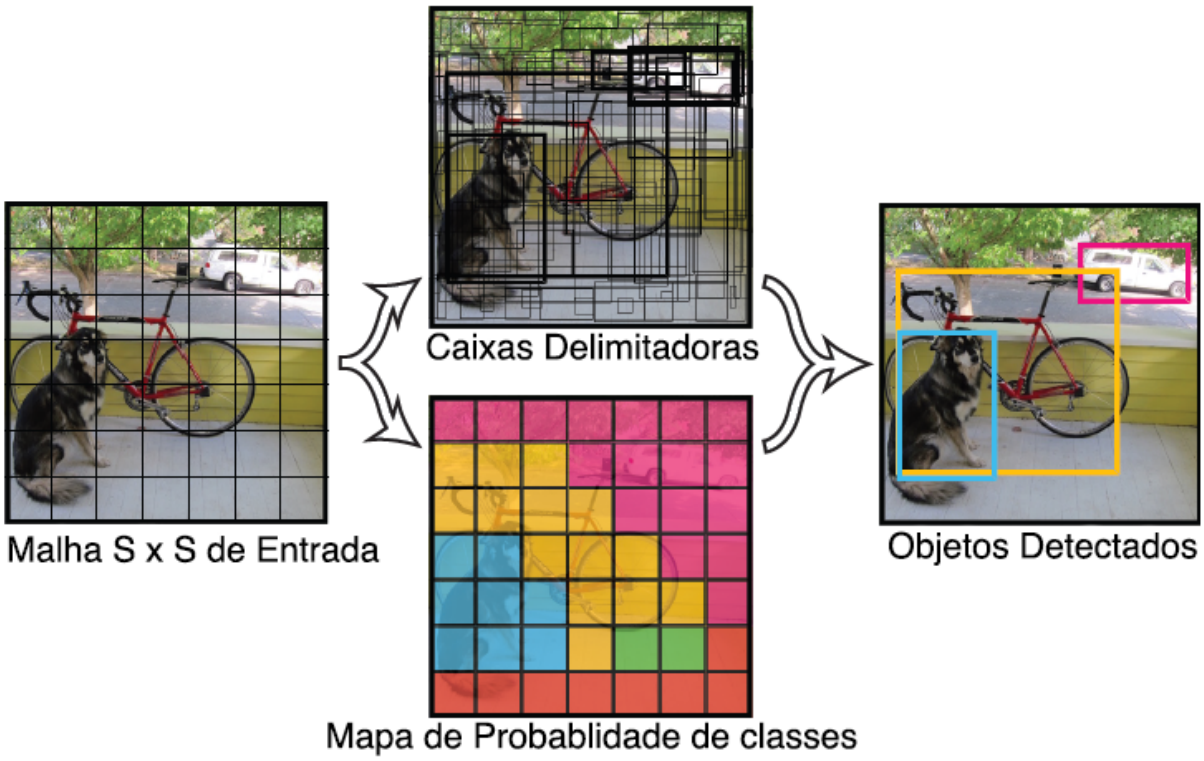
Uma vez que as caixas tenham sido obtidas, também são calculados parâmetros para cada uma delas, como o grau de confiança da existência de um objeto compreendido por elas. Também, é calculado o conjunto de probabilidades relativo à categoria do objeto compreendido pela célula, sendo que seu rótulo será dado pela maior probabilidade dentre as calculadas para cada categoria. Então, é feito o produto entre o grau de confiança de cada caixa, com a probabilidade da categoria mais provável da célula, resultando no grau de confiança da predição feita por cada

caixa. Embora sejam várias caixas delimitadoras por célula, o conjunto de probabilidade é o mesmo para todas as caixas pertencentes a uma mesma célula (20).

Como cada caixa pode se sobrepor a outras, e delimitar o mesmo objeto, é calculado o quanto cada uma se sobrepõem uma à outra, resultando em um parâmetro conhecido como valor da interseção sobre a união (IoU), sendo que ele é utilizado juntamente com a confiança de predição de cada caixa para eliminar aquelas que tenham menores valores e se sobreponham significativamente a outra caixa com elevado valor de confiança de predição. Esse processo é conhecido como algoritmo de Supressão das Não Máximas (20).

Na Figura 2.10, o funcionamento do YOLO é exemplificado, em que diversas caixas com diferentes dimensões estão se sobrepondo umas às outras, mas somente aquelas que possuem maior grau de confiança para o mesmo objeto, permanecem nas detecções obtidas pelo YOLO, enquanto as outras são descartadas no processo descrito anteriormente, eliminando detecções que representem o mesmo objeto.

Figura 2.10 – Exemplo de Detecção de Objetos com o Algoritmo YOLO

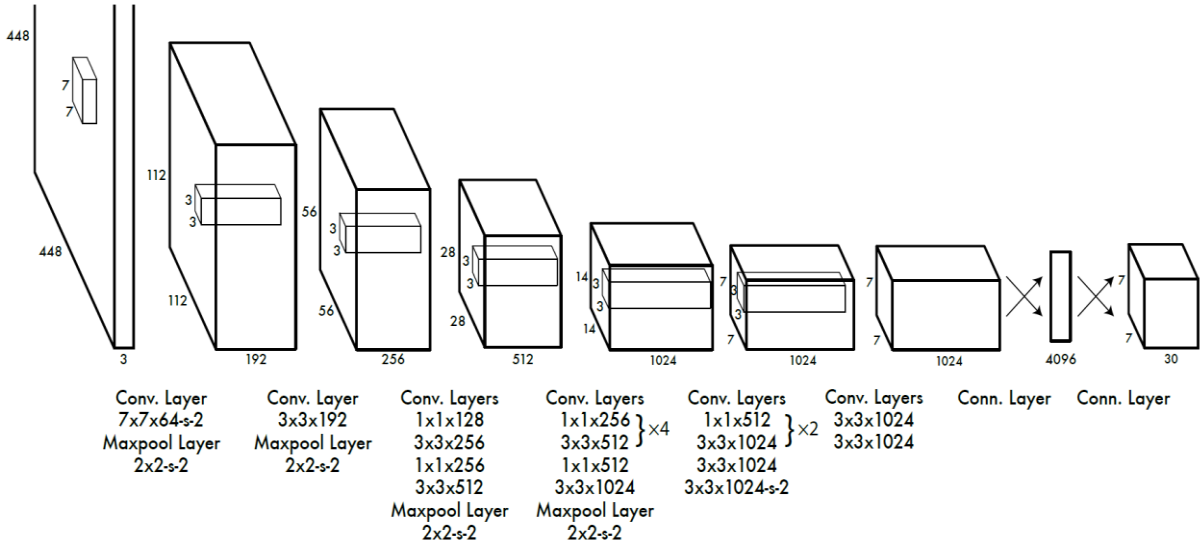


Fonte: Adaptado de (20).

Em termos de arquitetura utilizada, a rede neural construída no algoritmo original do YOLO, é composta por 24 camadas convolucionais, 4 camadas de *pooling* e 2 camadas totalmente conectadas entre si, como pode ser visto na Figura 2.11 (20).

O resultado de saída desta rede, por sua vez, é um tensor de dimensões $S \times S \times (B \cdot 5 + C)$, em que o termo $S \times S$ indica em quantas células a imagem de entrada foi dividida, B representa o número de caixas delimitadoras por célula, e C é o número de classes de objetos que a rede foi configurada para reconhecer (20). Para a rede utilizada na imagem da Figura 2.10, os autores utilizaram 49 células com 2 caixas delimitadoras e 20 classes, ou seja, $S = 7$, $B = 2$ e $C = 20$, resultando em uma rede com tensor de saída com o formato: $7 \times 7 \times 30$ (20), que codifica todas as informações de classe, posição, largura, altura e confiança de predição de cada caixa das células.

Figura 2.11 – Estrutura Original do Algoritmo YOLO



Fonte: (20).

Em versões mais recentes do algoritmo, para melhorar a capacidade de identificação de objetos na imagem, principalmente objetos pequenos, são utilizadas, em cada célula, as chamadas caixas âncora (22). Elas podem possuir localizações, formatos e tamanhos pré-definidos, sendo configuradas de forma a se especializarem em detectar objetos específicos (22), por exemplo, caixas âncora mais largas do que altas para reconhecer automóveis.

2.3 RECONHECIMENTO ÓPTICO DE CARACTERES (OCR)

É um campo dentro de visão computacional responsável pelo reconhecimento de caracteres presentes em uma imagem, sejam eles letras, números ou símbolos, convertendo-os em um formato compreensível para máquinas, com aplicações em digitalização de documentos, reconhecimento de placa veicular, dentre outras (23).

Os OCRs convencionais costumam se basear em cinco passos a serem executados: 1) aquisição de imagem; 2) pré-processamento de imagem; 3) segmentação de caracteres; 4) extração de características; 5) classificação de caracteres, podendo envolver uma etapa de pós-processamento em alguns casos (23).

2.3.1 Aquisição de Imagem

A etapa de aquisição de imagem se trata de digitalizar cenas estáticas com o auxílio de dispositivos, como câmeras, *scanners*, dentre outros tipos de dispositivos de captura de imagem, sendo uma etapa com grande responsabilidade sobre a precisão do OCR, visto que imagens de baixa qualidade, com caracteres distorcidos e presença de ruído, prejudicam o reconhecimento dos caracteres (23).

Para o reconhecimento de caracteres de placas de veículos, algumas relevantes fontes de degradação da imagem costumam ser desfoque de câmera e desfoque de movimento. O desfoque de câmera é causado pelas condições ambientais referentes ao local em que a imagem é obtida, enquanto o desfoque de movimento é causado devido ao tempo de exposição da imagem, visto que o objeto fotografado, por estar em alta velocidade, se move durante o tempo em que a imagem está sendo produzida, distorcendo-a (24). A Figura 2.12 mostra uma foto de um carro parado e outra do carro em movimento, podendo ser visto que a leitura dos caracteres da placa se torna mais difícil na imagem em que o carro estava se movimentando quando a imagem foi capturada, ilustrando os efeitos do desfoque de movimento.

Figura 2.12 – Exemplo de Degradação Causada pelo Desfoque de Movimento

Fonte: (24).

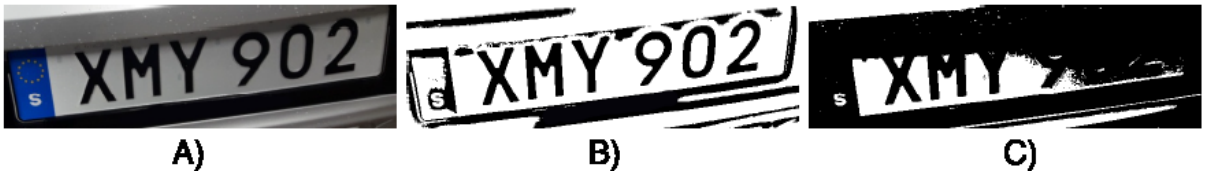
2.3.2 Pré-processamento de Imagem

Nessa fase, a imagem é tratada por filtros para redução de ruído, binarização, correção de inclinação e tratamento de outras características da imagem que possam prejudicar o reconhecimento de caracteres. Essa fase antecede outros estágios de processamento da imagem e tem o intuito de aprimorar os resultados obtidos pelas próximas fases (23).

Em outros trabalhos realizados visando localizar e reconhecer placas veiculares, como em (25), as medidas de pré-processamento incluíram a conversão da imagem do espaço de cores *Red*, *Green*, *Blue* ou Vermelho, Verde, Azul (RGB) para a escala de cor cinza ou *grayscale* (25). Essa é uma estratégia que pode ser utilizada tanto com o intuito de reduzir a carga computacional para processar a imagem, visto que este processo reduz a quantidade de canais necessários para representar a imagem de três para um, quanto para preparar a foto para passar por outros algoritmos, que dependem da representação da imagem em tons de cinza para funcionar adequadamente. Também, foi realizado uso do filtro gaussiano para a remoção de ruído, e uso do algoritmo de binarização para converter os *pixels* da imagem para preto e branco, utilizando um valor de limiar ou intervalo de valores (25).

A binarização também pode ser feita utilizando limiar adaptativo. Com esta técnica, um *kernel* de tamanho $N \times N$ é utilizado para percorrer a imagem, aumentando o valor do limiar para áreas com brilho elevado e diminuindo para regiões escuras (11). Dessa forma, o limiar adaptativo contorna uma dificuldade relativa à binarização realizada com limiar fixo, que é lidar com imagens que possuem regiões com diferentes luminosidades, visto que a binarização feita dessa forma causa perda de detalhes em imagens nestas condições (11). A Figura 2.13 ilustra a diferença entre estes métodos de binarização. Em A) é a imagem da placa original que, por conta da iluminação do ambiente, está com maior luminosidade à esquerda do que à direita, em B) há a imagem binarizada com limiar adaptativo, em que é possível notar que os caracteres constituintes ficaram devidamente visíveis, e em C) há a imagem binarizada com limiar fixo, podendo ser vista perda significativa de detalhes dos últimos três caracteres, que estavam na região com iluminação mais escura, tornando-os difíceis de serem reconhecidos.

Figura 2.13 – Exemplo de Binarização com Limiar Fixo e Limiar Adaptativo



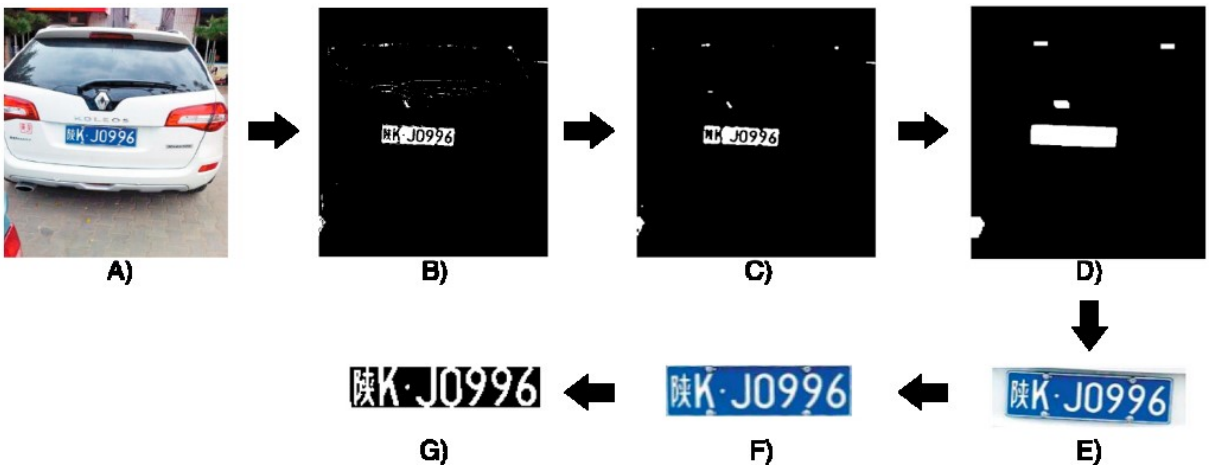
Fonte: Adaptado de: (11).

A conversão de espaço de cores ainda pode ser feita do RGB para *hue*, *saturation*, *value* (HSV), em que a imagem, embora continue sendo representada por três canais, é caracterizada pelo matiz, saturação e o valor de cada *pixel*. Como visto em outros trabalhos, esta representação permite realizar a binarização da imagem sem a necessidade de realizar a conversão para a escala de cinza (26).

Em termos de tratamentos morfológicos, alguns autores utilizam técnicas de erosão e dilatação, visando conectar áreas isoladas e remover pontos contínuos de ruído na imagem binarizada. Assim, a imagem resultante pode passar por filtros de detecção de borda para identificar contornos de objetos na imagem (26).

A seguir, a Figura 2.14 ilustra todas as etapas de pré-processamento e segmentação ocorridas para a imagem de um carro, visando identificar a região contendo a placa veicular para, então, recortá-la. Nesse caso, em A) é mostrada a imagem original, que passa por um processo de binarização, resultando na imagem em B). Então, a figura resultante passa pelo filtro da mediana no intuito de remover ruído, obtendo C). Utilizando tratamentos morfológicos, é obtido D). Então, a imagem é analisada para extrair a região que mais se assemelha à área de uma placa veicular. Para isso, a relação largura e altura dos objetos brancos é utilizada para identificar qual objeto tem o formato de uma placa para, então, ser recortado. Feito isso, a placa é extraída da imagem original, estando inclinada, como pode ser visto em E). Para corrigir a inclinação, é utilizado a Transformada de Hough para detectar bordas do objeto, possibilitando o cálculo do ângulo entre as bordas inferiores e superiores com a horizontal, o que é usado para remover a inclinação, resultando em F). Por fim, como pode ser visto em G), a placa é binarizada e, utilizando o método da projeção, são calculados quantos *pixels* não nulos (brancos) cada linha e coluna da imagem possui, identificando a linha superior e inferior que delimita a placa veicular para, então, remover as bordas da imagem não pertencentes à placa.

Figura 2.14 – Exemplo de Aplicação de Algoritmos de Pré-processamento



Fonte: Adaptado de: (26).

2.3.3 Segmentação de Caracteres

Nessa fase, a imagem pré-processada tem seu conteúdo dividido ou segmentado em partes menores no intuito de facilitar e melhorar o processamento da imagem nas etapas posteriores. Para isso, a segmentação pode ser feita dividindo o conteúdo em parágrafos, linhas, palavras, caracteres e sub-caracteres. Então, cada parte segmentada da imagem é levada para a etapa de extração de características e classificação (23).

A segmentação pode ser feita utilizando o método da projeção vertical, que consiste em contar o número de *pixels* não nulos (brancos) em cada coluna da imagem binarizada. Como há uma diferença entre o número de *pixels* brancos em uma coluna contendo parte de um caractere, com uma coluna que não contém, é possível determinar regiões contendo caracteres em uma imagem utilizando esta técnica (26). A Figura 2.15 ilustra a segmentação realizada na placa da Figura 2.14 utilizando este método.

Figura 2.15 – Placa Veicular Segmentada

Fonte: (26).

Como se nota pela Figura 2.15, os vãos entre os caracteres foram eliminados, bem como o ponto que separa os dois primeiros caracteres do restante. Isso foi possível pois, como colunas que não possuem *pixels* pertinentes a partes de caracteres apresentam uma quantidade menor de *pixels* brancos, elas foram removidas. Feita a segmentação, cada caractere é passado para o próximo estágio do processo de reconhecimento de caracteres.

2.3.4 Extração de Características

Na extração de características, é feita a coleta de padrões únicos de cada segmento da imagem, no intuito de obter características que diferenciem um caractere do outro, e de reduzir a quantidade de dados utilizados para fazer o reconhecimento (23).

As características extraídas podem ser oriundas de transformações globais, estruturais e estatísticas. A primeira se refere a características obtidas de transformações, também utilizadas em segmentação e outros estágios, como projeção horizontal e vertical. A segunda é pertinente à topologia e estrutura dos caracteres, como pontos de intersecção, linhas horizontais e verticais, largura, tamanho, dentre outras características. A terceira, por sua vez, refere-se a medidas quantitativas, como distância e números de *pixels* (23).

Em trabalhos utilizando redes neurais convolucionais para reconhecimento de caracteres não há a necessidade de emprego de métodos manuais de extração de características, como os mencionados anteriormente, pois a própria estrutura da rede faz isso por meio dos filtros das camadas convolucionais (24). Ademais, como os filtros possuem seus valores ajustados de forma automática durante o processo de treinamento da CNN (19), não há a necessidade de realizar o ajuste manual destes filtros.

2.3.5 Classificação de Caracteres

O estágio de classificação de caracteres recebe a imagem vinda das etapas anteriores e é responsável por decidir quais são os caracteres presentes. Para isso, diferentes classificadores podem ser utilizados, como aqueles baseados em redes

neurais artificiais, máquina de vetores de suporte ou *Support Vector Machine* (SVM), dentre outros (23).

Para os classificadores baseados em redes neurais artificiais, é comum o uso das redes neurais convolucionais pelos motivos já apresentados, como eficiência computacional, dispensa de implementação das etapas de pré-processamento e extração de características, dentre outras vantagens. Apesar de ser menos utilizada para esta finalidade, a rede neural multicamadas, como a rede da Figura 2.5, também pode ser empregada em classificadores (23).

Embora a SVM seja um algoritmo originalmente desenvolvido para classificação binária, esta tecnologia pode ser adaptada para a classificação de mais de duas classes, viabilizando o reconhecimento de caracteres, como utilizando em (26), em que caracteres segmentados, como os da Figura 2.15, foram identificados por uma SVM. Essa ferramenta tem, como princípio, fazer o mapeamento dos dados de entrada em um espaço de maior dimensão para, então, obter o hiperplano que separa as diferentes classes de dados, sendo o hiperplano ótimo aquele que maximiza a distância entre ele e cada uma das classes (23).

Dentre as abordagens de reconhecimento de caracteres não baseadas em aprendizado de máquina, há o método da comparação. A base desta metodologia é segmentar cada caractere em partes menores, chamadas de sub-caracteres, para, então, após binarizar cada imagem resultante, compará-los com sub-caracteres pertencentes a caracteres conhecidos (25). Como a diferença entre imagens que representem partes dos mesmos caracteres será pequena, o método consegue discernir quais são os caracteres baseado na semelhança entre os sub-caracteres.

2.4 PLACAS VEICULARES BRASILEIRAS

Outro aspecto importante relativo à detecção e reconhecimento de caracteres de placas veiculares são as características físicas da própria placa veicular.

No Brasil, as placas de veículos de quatro rodas ou mais rodas em geral, como carros e ônibus, possuem apenas uma linha de caracteres, enquanto as placas de veículos de duas rodas, como motos e ciclomotores, geralmente, possuem duas linhas contendo caracteres, além de serem placas com menores dimensões e espaçamento entre caracteres. Dessa forma, tais fatores impõem dificuldades extras nos diferentes

estágios dos sistemas ALPR de modo que em muitas pesquisas envolvendo detecção e reconhecimento de caracteres de placas veiculares, imagens de veículos com essas particularidades são descartadas. A Figura 2.16 ilustra essa diferença entre placas, com a placa vermelha pertencente a um caminhão, e as outras duas pertencentes a motos.

Figura 2.16 – Exemplos de Placas Veiculares Brasileiras

Fonte: (27).

Outro fator relevante é que em 2020, o Brasil adotou um novo modelo de placa, chamado de modelo Mercosul, de forma que, nos dias atuais, coexistem veículos com placas veiculares antigas, como as da Figura 2.16, e com as placas Mercosul, sendo elas esteticamente diferentes uma da outra e com uma diferença no padrão de caracteres. Enquanto a placa Mercosul possui três letras seguidas de um número, outra letra e mais dois números, a placa antiga possui três letras seguidas de quatro algarismos (28). Na Figura 2.17 é possível ver um exemplo de placa pertencente ao padrão mais recente, com as diferenças mencionadas:

Figura 2.17 – Exemplo de Placa Padrão Mercosul

Fonte: (28).

Dessa forma, tais diferenças devem ser consideradas no desenvolvimento de um Sistema ALPR pois, por exemplo, caso somente a placa antiga estivesse em uso, um algoritmo de pós-processamento poderia ser utilizado para reconhecer os quatro últimos caracteres sempre como números. Assim, por exemplo, caso fosse reconhecido a letra “O” em algum deles, por engano, o sistema definiria o caractere reconhecido como o número 0, por ser o algarismo mais similar com letra “O”, assim como feito em outros trabalhos (29). Entretanto, como a placa Mercosul possui uma

letra a mais do que o modelo antigo, isto reduz as possibilidades de pós-processamento para aumentar a acurácia do sistema, ou aumenta sua complexidade, visto que seria necessário adicionar a capacidade de discernir qual é o modelo de cada placa.

Por fim, para algoritmos baseados em aprendizado de máquina, a coleta de imagens de placas pode representar um desafio em termos de diversidade de dados visto que, em cada Estado brasileiro, há um conjunto específico de letras que podem ser usadas, o que pode gerar vieses no algoritmo, reconhecendo corretamente certas letras mais do que outras (27). Por exemplo, em placas veiculares originárias do Paraná, os três primeiros caracteres que podem ser utilizados estão compreendidos de AAA até BEZ (29). Logo, alguns caracteres, como as letras A e B, são mais recorrentes em placas deste Estado, enquanto outros, como a letra P, não são.

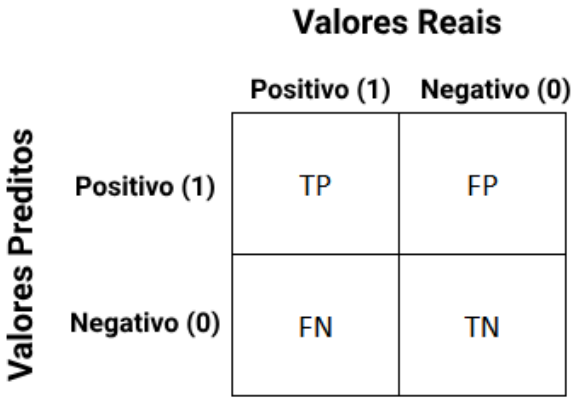
2.5 PARÂMETROS DE AVALIAÇÃO DE DESEMPENHO DE UMA REDE NEURAL ARTIFICIAL

Existem diversos parâmetros que podem ser analisados para avaliar a performance de um modelo treinado. Cada um deles traz informações mais relevantes para avaliar a performance do modelo obtido dependendo do tipo de problema tratado. A seguir, alguns dos parâmetros pertinentes a modelos de visão computacional serão apresentados.

2.5.1 Matriz Confusão

Para problemas envolvendo classificação, em que os sinais de entrada são mapeados em diferentes classes na saída da rede, um dos métodos de avaliação que podem ser utilizados é a Matriz Confusão (*Confusion Matrix*). Exemplificado pela Figura 2.18, este parâmetro permite analisar a performance do modelo para cada classe individualmente, revelando se a rede neural treinada é capaz de reconhecer uma classe melhor do que outra, se o modelo possui tendências de reconhecer erroneamente objetos de uma determinada classe como pertencentes a outra, dentre outros aspectos que a matriz confusão permite visualizar (30).

Figura 2.18 – Exemplo de Matriz Confusão



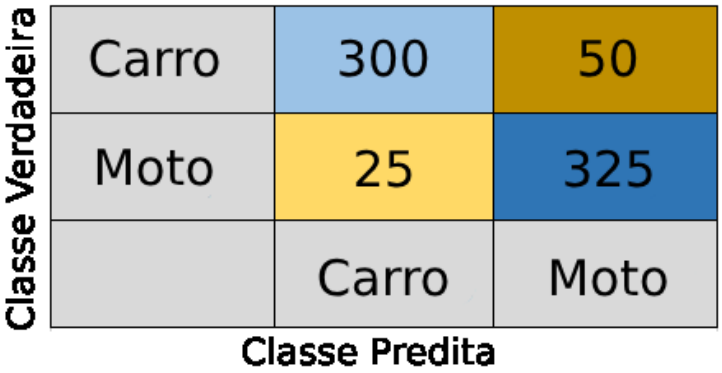
Fonte: Adaptado de: (31).

A Matriz Confusão, como pode ser visto na Figura anterior, relaciona quatro valores para uma determinada classe. O TP ou Verdadeiros Positivos (*True Positive*) indica quantas amostras de entrada pertenciam a esta classe e foram corretamente classificadas pelo modelo. FP ou Falsos Positivos (*False Positive*) representa quantos sinais de entrada não pertencentes àquela classe foram erroneamente classificados como tal. TN ou Negativos Verdadeiros (*True Negative*) aponta quantas amostras que não pertenciam a determinada classe não foram classificadas pelo modelo como pertencendo a esta classe. Por fim, FN ou Falsos Negativos (*False Negative*) indica quantas vezes o modelo deixou de classificar corretamente um sinal de entrada que pertencia à classe em análise (31).

A Matriz Confusão também pode ser representada relacionando as classificações feitas pela rede para cada classe. Por exemplo, para um problema binário, como o da Figura 2.19, uma rede neural pode ser utilizada como base para realizar o reconhecimento do tipo de veículo presente em uma imagem, avaliando se a imagem de entrada é referente a um carro ou uma moto. No eixo vertical, há as classes verdadeiras, enquanto no eixo horizontal, as classificações feitas pelo modelo em questão. Neste caso, em 300 imagens em que a classificação correta era “carro”, o modelo fictício classificou corretamente. No entanto, em 50 ocasiões, a resposta correta era “carro”, mas foi obtido “moto”. Nesse mesmo sentido, em 325 vezes, a rede classificou imagens de motos como pertencentes à classe “moto”. Porém, em 25 imagens, a resposta era “moto”, mas a previsão entregue pelo modelo foi “carro”.

Da mesma forma que a Matriz da Figura 2.18 permite obter o TP, FP, FN e TN, a Matriz da Figura 2.19 também permite obter estes parâmetros para cada classe. Neste caso, o parâmetro Verdadeiros Positivos para a classe “carro” é 300, enquanto os Falsos positivos são 25, os Falsos Negativos são 50 e, por fim, os Negativos Verdadeiros são 325.

Figura 2.19 – Matriz Confusão para um Caso de Classificação de Veículos



Fonte: Autor.

Além disso, há problemas em que a amostra de entrada da rede pode não assumir nenhuma das classes que a rede neural foi treinada para reconhecer. Por exemplo, para o caso fictício da Figura 2.19, no conjunto de imagens de entrada poderiam haver casos de fotos sem a presença de um carro ou de uma moto, ou contendo um ônibus ou uma bicicleta. Então, o modelo treinado poderia categorizar erroneamente algum objeto destas imagens como pertencendo à classe “carro” ou “moto”. Cabe destacar que a Matriz Confusão não se limita a problemas binários, podendo ser utilizada, também, em problemas com múltiplas classes (30). Neste caso, a matriz da Figura 2.19 poderia assumir múltiplas colunas e linhas referente às diversas classes que um modelo treinado poderia reconhecer.

2.5.2 *Recall*

Utilizando os quatro parâmetros mencionados, é possível calcular outros, como o *Recall*, que é dado pela razão dos Verdadeiros Positivos pela soma dos Verdadeiros Positivos com os Falsos Negativos, conforme apresentado pela equação (2.4), e representa o quanto o modelo foi capaz de classificar corretamente uma classe

quando uma amostra desta mesma classe foi apresentada ao modelo. Ademais, este parâmetro é independentemente do quão bem ou ruim foram os resultados da rede neural para amostras de outras classes (30). Isto é, o *recall* mede o quão bem o modelo reconheceu amostras de determinada classe e, tipicamente, os valores de *recall* são diferentes para cada classe. Para um caso de identificação de veículos, como o da Figura 2.19, o *recall* poderia ser utilizado para verificar o quão bem ele classifica imagens de carros como pertencente à classe “carro”.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

2.5.3 Acurácia

Outro parâmetro é a acurácia, que permite quantificar a performance do modelo levando em consideração todas as classes. A acurácia é dada pela equação (2.5), que relaciona os Verdadeiros Positivos e Negativos, com os Falsos Positivos e Negativos, isto é, a acurácia é dada pela razão entre o número de predições corretas para uma classe pelo número total de predições (30).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

2.5.4 Precisão

A precisão, por sua vez, mede a confiança do modelo em classificar uma amostra como pertencente a determinada classe. Ela é dada pela razão dos Verdadeiros Positivos pela soma dos Verdadeiros Positivos com Falsos Positivos, (30), conforme equação (2.6). Ou seja, a precisão somente é elevada se o modelo classificar corretamente as amostras desta classe. Se, além disso, amostras de outras classes também forem classificadas como pertencentes a esta classe, então, a precisão será baixa, ou seja, a confiança no modelo para rotular uma amostra como sendo de determinada classe, será pequena (30). Por exemplo, se um modelo de identificação de veículos classifica todas as imagens de motos como pertencentes à

classe “moto”, mas, também, classifica todas as imagens de bicicletas como pertencentes à classe “moto”, então, a precisão deverá possuir um valor baixo.

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

2.5.5 F1 *Score*

Para comparar modelos com alta precisão e baixo *recall* ou vice-versa, há o parâmetro *F-Score* ou F1 *Score*, que realiza a média harmônica entre ambos os parâmetros, conforme pode ser visto pela equação (2.7), encapsulando a performance da rede em um único parâmetro (31).

$$F_{Score} = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{2.7}$$

Tipicamente, o *F-Score* é utilizado para problemas em que o *recall* e a precisão são igualmente importantes (32).

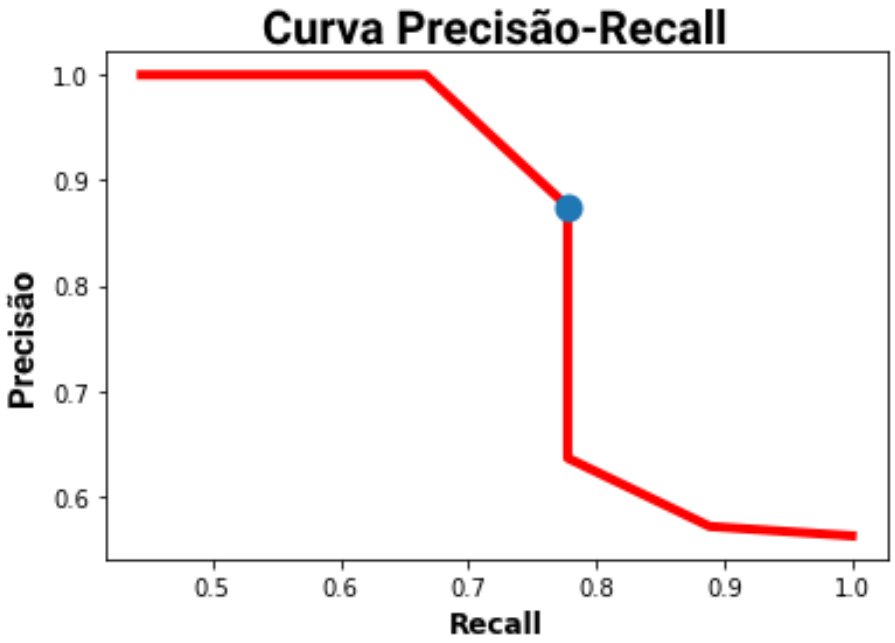
2.5.6 Curva PR

Modelos de classificação de objetos são caracterizados por apresentarem em sua saída valores que representam a classificação de um sinal que tenha sido utilizado como entrada do modelo. Para definir a classe pertencente ao sinal de entrada, são utilizados *thresholds* ou limiares. Se o valor do sinal de saída for maior, menor ou igual ao *threshold*, o sinal de entrada irá pertencer a determinada classe ou não (33).

Neste contexto, é possível calcular a curva PR ou curva precisão-*recall*, que é obtida calculando os valores de precisão e *recall* para diferentes valores de limiares. Assim, é possível selecionar o *threshold* que maximiza ambos os valores. Para encontrar este limiar também pode ser utilizado o F1 *Score*. Ao calcular este parâmetro para cada par de valores de *recall* e precisão, o maior valor encontrado para o F1 *Score* será aquele pertinente ao limiar que maximiza *recall* e precisão (33). A seguir, na Figura 2.20, é possível ver um exemplo de curva PR. Como é possível

notar, maior o *recall*, menor a precisão e vice-versa, no entanto, é possível escolher um ponto da curva que maximiza ambos parâmetros.

Figura 2.20 – Exemplo de Curva PR



Fonte: Adaptado de: (33).

2.5.7 *Mean Average Precision* (mAP)

O parâmetro *Mean Average Precision* (mAP) é tipicamente utilizado para avaliar a performance de modelos de detecção de objetos em imagens, como o YOLO. Para o cálculo desta métrica, são utilizadas as coordenadas verdadeiras do objeto a ser detectado, as coordenadas obtidas pelo modelo treinado para realizar o reconhecimento, o *recall* e a precisão. Em resumo, quanto maior for o mAP, maior é a acurácia do modelo em detectar objetos (33).

Nesse sentido, para obter esta métrica, primeiro é obtido a Precisão ponderada (*Average Precision* ou AP), que é um valor que descreve a média de todas as precisões para uma determinada classe de objeto, ou seja, descreve a curva PR em um único valor. O cálculo deste parâmetro é dado pela equação a seguir, em que o AP é dado pela soma ponderada das precisões para cada valor de *threshold*. Ademais, para efeitos de cálculo, é considerado para os valores de *Recall* e Precisão,

as seguintes condições: $Recall(n) = 0$ e $Precision(n) = 1$, sendo n o número de *thresholds* (33).

$$AP = \sum_{k=0}^{k=n-1} [Recall(k) - Recall(k+1)] \cdot Precision(k) \quad (2.8)$$

Dessa forma, a equação (2.8) revela que parâmetro AP é a área abaixo da curva PR, e é um valor que varia entre 0 e 1, visto que o *recall* e a precisão podem assumir somente valores dentro deste intervalo.

Ademais, o número de limiares é arbitrário. Por exemplo, para calcular o parâmetro AP@[.5:.95], que é o AP para valores de limiar entre 0,5 e 0,95, espaçados por um fator de 0,05, são utilizados dez valores de *threshold* (34).

Para problemas de detecção de objetos, o limiar representa o valor que a interseção entre a caixa delimitadora verdadeira e a caixa delimitadora obtida pelo modelo treinado precisa ter para que seja considerado que a detecção de um objeto presente na imagem tenha sido realizada com sucesso. Por exemplo, se um sistema de detecção de carros realiza uma detecção em que a caixa delimitadora obtida compreende cerca de 0,75 ou 75% da região da imagem que contém o carro, para *thresholds* superiores a 0,75, esta detecção será considerada como uma falha, enquanto *thresholds* menores do que 0,75 consideram esta detecção como um sucesso.

Dessa forma, para calcular o mAP para um modelo com n_c classes, utilizando os limiares mencionados, basta calcular o AP para cada classe, somar os resultados e dividir pelo número de classes, conforme equação (2.9) demonstra (33). Portanto, o mAP pode ser sintetizado como a média da precisão ponderada obtida para cada classe que a rede neural foi treinada para reconhecer.

$$mAP = \frac{1}{n_c} \cdot \sum_{k=1}^{k=n_c} AP_k \quad (2.9)$$

Por exemplo, para o caso fictício da Figura 2.19, no intuito de calcular o mAP considerando uma faixa de *thresholds* indo de 0,5 até 0,95, ou seja, calculando o mAP@[.5:.95], primeiro é deverá ser calculado, o *Recall* e a Precisão para cada uma

das duas classes considerando cada um dos valores de *threshold*. Então, a precisão ponderada ou AP@[.5:.95] é calculada, realizando a média ponderada da precisão. Após isso, o mAP é obtido realizando a média aritmética do AP considerando todas as classes do problema.

2.6 CONSIDERAÇÕES FINAIS

Nesse capítulo, foi apresentado o referencial teórico deste projeto, com grande ênfase nas técnicas de visão computacional, tanto baseadas em aprendizado de máquina quanto em processamento digital de imagens. Também, foram evidenciados os parâmetros de avaliação dos modelos treinados por estas técnicas para ser possível avaliar, ao final deste trabalho, a performance dos modelos treinados. Por fim, foram ressaltadas as particularidades relevantes das placas veiculares brasileiras e como elas podem influenciar o projeto. No próximo capítulo, serão discutidos aspectos de metodologia e recursos necessários para a execução deste trabalho.

CAPÍTULO 3

MATERIAIS E MÉTODOS

Neste capítulo, serão apresentados os recursos necessários para este projeto, e os métodos empregados para sua execução.

3.1 MÉTODOS

Este projeto foi desenvolvido em três grandes etapas, descritas nas próximas seções, sendo elas: 1) Reconhecimento de Placas; 2) Reconhecimento de Caracteres; 3) Integração com *smartphones* Android.

Para o desenvolvimento de cada etapa, a pesquisa realizada teve caráter tanto bibliográfico quanto experimental, baseada em artigos, livros, videoaulas, exemplos de códigos, dentre outras fontes de conhecimento, para a definição das melhores e mais viáveis tecnologias para a implementação de cada etapa mencionada anteriormente, sendo realizados testes de cada tecnologia utilizando um computador e um *smartphone*.

Dessa forma, os objetos de estudo deste trabalho foram as ferramentas de visão computacional e processamento digital de imagens, tendo um pouco de ênfase em outras ferramentas secundárias, como as plataformas de desenvolvimento *mobile*.

3.1.1 Reconhecimento de Placas

Nesta etapa, o *software* foi desenvolvido para ser capaz de examinar uma imagem, encontrar placas veiculares, caso existam, e recortá-las para que seja feito o reconhecimento de caracteres.

Para isso, o sistema proposto realiza a coleta imagens após o usuário pressionar o botão na aplicação Android. Então, para analisar cada imagem coletada em busca da região da placa, foi empregado o algoritmo YOLO, por se tratar de um algoritmo de código aberto, e ser considerado o “Estado da Arte” em detecção de objetos (29). Nesse caso, diferentemente de projetos desenvolvidos em outros trabalhos (4), não foi implementada uma etapa de detecção de veículos, visto que o

algoritmo se mostrou bastante capaz de localizar as placas sem a necessidade de localizar o veículo primeiro.

Dado que o treino da rede neural é uma etapa mais complexa, foi realizado um estudo sobre como e quais ferramentas utilizar para isso. Dessa forma, foi utilizado um sistema de computação em nuvem e um *software* de cálculo numérico para realizar o preparo dos dados do banco de imagens e treinar o modelo.

Após a aplicação conseguir coletar a imagem, localizar a placa veicular e recortá-la, o sistema inicia a etapa de reconhecimento de caracteres.

3.1.2 Reconhecimento de Caracteres

Para esta etapa, primeiro, foi treinado outro modelo do algoritmo YOLO, para realizar a segmentação dos caracteres, visto que esta tecnologia é capaz não somente de detectar objetos, mas obter suas localizações na imagem. Dessa forma, é possível recortar os objetos para, então, enviá-los para os modelos que realizam o reconhecimento de cada caractere.

Considerando que o YOLO é limitado quanto à identificação de objetos pequenos, duas redes neurais convolucionais foram implementadas: uma dedicada ao reconhecimento de letras e outra dedicada ao reconhecimento de números. Para isso, os modelos foram treinados utilizando computação em nuvem e imagens do banco mencionado anteriormente.

3.1.3 Integração com smartphones Android

Nesta etapa, foi construído uma aplicação Android para receber os modelos de detecção de placa veicular, segmentação e reconhecimento de caracteres, e, dado uma imagem contendo um veículo, realizar o reconhecimento dos caracteres constituintes da placa veicular presente na imagem. Para isso, o *software* foi construído em uma plataforma de desenvolvimento *mobile* e testada no *smartphone* do autor.

3.2 MATERIAIS

Em termos de recursos que foram necessários para a execução do projeto, destacam-se:

a) *Smartphone* de capacidade computacional intermediária, necessário para executar e testar a aplicação;
b) Computador com *softwares* de desenvolvimento de aplicações Android e de tratamento e análise de dados;
c) Serviço de Computação em Nuvem para o desenvolvimento dos modelos de inteligência artificial empregados no reconhecimento de placas, segmentação e reconhecimento de caracteres.

O *Smartphone* utilizado, que é o principal recurso para poder avaliar a performance do sistema desenvolvido e servir de referência, foi o Samsung A32.

A ferramenta de desenvolvimento da aplicação *mobile* utilizada foi o Android Studio, que é um *software* gratuito e voltado para o desenvolvimento de aplicações para a plataforma Android (35). Também, para construir a aplicação, foi empregado a linguagem de programação Java, junto com a biblioteca Chaquopy, que viabiliza o uso da linguagem Python dentro da plataforma Android (36). Para implementar as redes neurais treinadas, foi utilizada a biblioteca TensorFlow Lite, que disponibiliza ferramentas para executar modelos treinados em dispositivos *mobile*, como *smartphones* Android, e dispositivos de borda, como microcontroladores (37).

Para treinar as redes neurais utilizadas, foi utilizado o banco de imagens de veículos conhecido como UFPR-ALPR, que é uma base de imagens pública, cedida pela Universidade Federal do Paraná (UFPR), construída com imagens de veículos do Estado do Paraná, no Brasil, com 4.500 imagens, sendo 150 veículos fotografados em 30 momentos diferentes. Uma das vantagens desta base de imagens é que ela acompanha as anotações das imagens, contendo a posição do carro, a posição da placa, os caracteres constituintes, dentre outras informações (29).

Para preparar as imagens e seus rótulos, a plataforma Matlab foi utilizada, visto que é uma ferramenta com diversas bibliotecas e funções integradas para a realização

de cálculos envolvendo matrizes (38), que é a forma como os dados das imagens utilizadas estão organizados.

Para construir e treinar o YOLO e as Redes Neurais Convolucionais, foi utilizado o Google Colab, que é um ambiente de computação em nuvem da Google, que oferece recursos computacionais, como *Graphics Processing Units* (GPUs) ou Unidades de Processamento Gráfico, em tradução livre, de forma gratuita, porém, com certas limitações (39). Para usar o Google Colab, foi feito uso da linguagem de programação Python. Além disso, foi utilizado o repositório do YOLOv5 (40), que contém os recursos necessários para treinar o YOLO, como a função utilizada para treinar os modelos.

Por fim, em termos de recursos financeiros, não houve despesas, visto que os materiais utilizados ou o autor já possuía acesso prévio, ou são gratuitos.

CAPÍTULO 4

DESENVOLVIMENTO

Neste capítulo, será apresentado o desenvolvimento deste trabalho, ressaltando como os modelos de reconhecimento foram construídos, bem como foi realizada a implementação da aplicação de reconhecimento de placas veiculares em um sistema Android. Por fim, será apresentado um exemplo de caso de uso da aplicação desenvolvida.

4.1 RECONHECIMENTO DE PLACAS

Para preparar o modelo de reconhecimento de placas, as imagens e suas anotações passaram por alguns ajustes para, posteriormente, o treinamento do modelo ser realizado. Nas próximas seções, estes aspectos serão apresentados.

4.1.1 Preparação dos Dados para o Reconhecimento de Placas

Para treinar, validar e testar o YOLO, o primeiro passo realizado foi o preparo das imagens e rótulos do banco de dados utilizado. Dessa forma, visando reduzir a carga computacional para executar e treinar o modelo, as fotos originais, que possuem 1920 por 1080 *pixels*, foram recortadas, resultando em imagens com 1056 por 1056 *pixels*. A escolha por estas dimensões foi realizada considerando que o YOLO utiliza imagens com dimensões iguais e múltiplas de 32 durante o processo de treinamento e execução.

Ademais, como o sistema é dedicado para o reconhecimento de placas com somente uma linha de caracteres, como é o caso de placas veiculares de carros, caminhões, ônibus, dentre outros tipos de veículos, foi realizado o descarte de imagens de veículos que não se enquadram neste critério.

Estas ações foram realizadas utilizando algoritmos desenvolvidos utilizando a plataforma Matlab (38) e o Google Colab (39). O código utilizado juntamente com outros *scripts* desenvolvidos ao longo deste trabalho, podem ser consultados em um repositório dentro da plataforma GitHub (41), criado especificamente para este projeto.

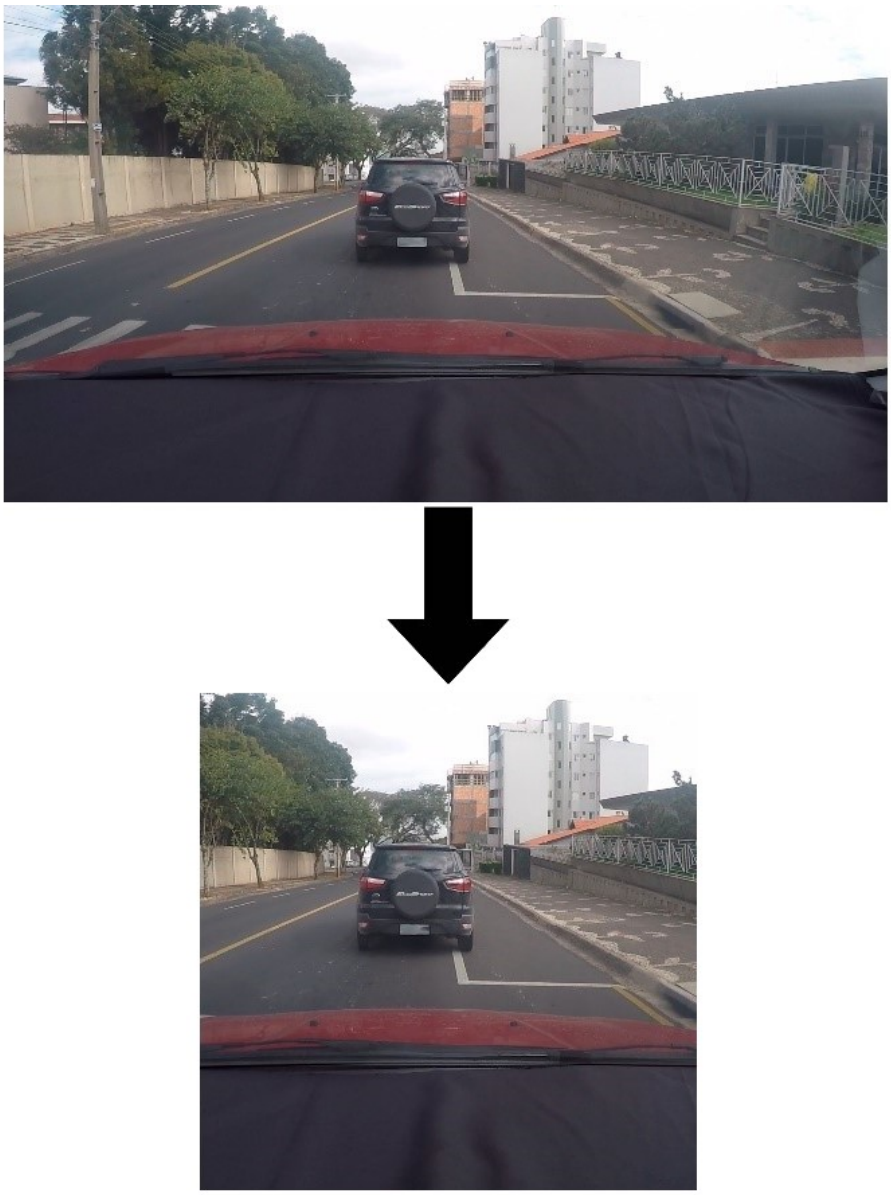
Após ajustar as imagens, foi realizado o preparo de seus rótulos, visto que o formato dos dados utilizados pelo YOLO é um molde específico em que, para cada objeto presente em uma determinada imagem que, no caso deste trabalho, são placas veiculares, o rótulo possui uma linha de texto contendo as seguintes informações: 1) um número representando o índice da classe do elemento; 2) Coordenada horizontal do centro do objeto; 3) Coordenada vertical; 4) Largura e, por fim, 5) Altura. Sendo todas as dimensões do objeto normalizadas pelo valor absoluto das dimensões da imagem (42).

Dessa forma, os rótulos foram convertidos para o formato necessário. Para esse fim, durante o cálculo das coordenadas no padrão YOLO, foi considerado as mudanças ocasionadas pela diminuição de *pixels* provocada pelo passo anterior.

Após ajustar as imagens e rótulos, como mencionado anteriormente, em algumas imagens, em que a placa veicular estava presente próximo à lateral esquerda ou direita da imagem, houve o recorte parcial ou total desta placa veicular. Assim, este processo de ajuste gerou algumas imagens sem a presença de uma placa veicular a ser detectada e reconhecida. Consequentemente, não foram gerados rótulos no formato YOLO para estas imagens, visto que não haviam placas a serem detectadas pelo modelo. Mesmo assim, estas fotos foram mantidas no conjunto de dados utilizado para o treinamento do modelo, visto que a existência de imagens sem rótulos é algo que não existe no conjunto de imagens original, mas é desejável, visto que isso contribui para reduzir falsos positivos (43), ou seja, melhora a capacidade de distinção do modelo, diminuindo as chances de outros objetos diferentes serem identificados erroneamente como placas.

A seguir, a Figura 4.21 ilustra um exemplo do tratamento dado às imagens, sendo o rótulo referente à placa veicular presente nesta foto igual a: 0 0.430871 0.475852 0.060606 0.019886, em que o primeiro termo, 0, representa a classe do objeto, e os demais valores são as coordenadas mencionadas anteriormente, normalizadas por 1056. Por questões de privacidade e segurança, os caracteres da placa veicular foram omitidos.

Figura 4.21 – Antes e Depois do Ajuste de Imagens



Fonte: Autor.

4.1.2 Treinamento do Modelo de Reconhecimento de Placas

Após preparar as imagens e rótulos, seguiu-se para a etapa de treinamento do modelo. Para isso, o conjunto de imagens, contendo 3.600 imagens, obtido após ajustes feitos na seção anterior, foi subdividido em 1.440 imagens para o treinamento, 720 para validação e 1.440 para testes.

Considerando que o YOLOv5 possui diferentes versões, com cada uma possuindo diferentes quantidades de parâmetros treináveis (40), e que o modelo será executado em um dispositivo de capacidade computacional limitada, foi escolhida a

menor versão do YOLO, também chamada de YOLOv5n ou YOLOv5 nano, abdicando de maior acurácia para obter um modelo que seja menos oneroso do ponto de vista computacional.

Nesse sentido, utilizando a plataforma de código aberto, Google Colab (39), foi realizado o treinamento do YOLOv5n. Para isso, foram definidas dez épocas de treinamento e oito imagens para cada *batch*. Assim, a rede teve seus pesos e limiares atualizados a cada oito imagens do conjunto de treinamento processadas pela rede neural, realizando este processo dez vezes.

A Figura 4.22 ilustra a detecção de placas realizada em diferentes imagens do conjunto de testes pelo modelo treinado. Para isso, foi desenhado um retângulo vermelho em torno dos objetos detectados utilizando as coordenadas fornecidas pela predição realizada pelo YOLOv5n.

Figura 4.22 – Exemplos de Placas Identificadas pelo Modelo Treinado

Fonte: Autor.

4.2 RECONHECIMENTO DE CARACTERES

Finalizada a etapa de reconhecimento de placas, o próximo passo foi desenvolver o reconhecimento de caracteres. Esta etapa foi subdividida em segmentação e identificação de caracteres, conforme apresentado nas próximas seções.

4.2.1 Pré-processamento das Imagens para a Segmentação de Caracteres

Para realizar a segmentação dos caracteres, foi escolhido a versão média do YOLOv5, ou YOLOv5m, tendo em vista sua melhor capacidade em detecção de objetos utilizando baixa carga computacional (11). Por ser uma versão maior do que a utilizada para o reconhecimento de placas, ela possui mais parâmetros para treinar, no entanto, como as imagens das placas são significativamente menores do que as imagens dos veículos utilizadas no reconhecimento de placas, o custo computacional de utilizar uma versão maior do YOLO é compensando pela menor quantidade de informações para processar em cada imagem.

Dessa forma, assim como foi necessário preparar os dados para o treinamento do modelo de reconhecimento de placas, também foi realizado o mesmo para os dados de treinamento do modelo de segmentação. Nesse sentido, foi feito a leitura dos rótulos originais para recortar as placas, redimensioná-las para 240 por 80 *pixels*, e preenchê-las com *pixels* pretos para obter figuras com 256 por 256 *pixels*. Assim, foi construído um novo banco de dados contendo apenas imagens de placas veiculares com dimensões padronizadas, múltiplas de 32 e iguais conforme o padrão de imagens utilizado para treinar o YOLOv5.

Finalizado o ajuste das imagens, foi realizado o preparo dos rótulos. Para isso, foi necessário considerar as alterações ocorridas nas coordenadas de cada caractere ocasionadas pelo redimensionamento das imagens.

Ademais, como a cor e outras características da placa veicular podem ser diferentes de acordo com a categoria do veículo (29), no intuito de aumentar o conjunto de treinamento e validação, foi realizada a inversão das cores de todas as placas da base de imagens, simulando placas de outras categorias. A Figura 4.23 ilustra algumas imagens de placas com cores invertidas e não invertidas, mostrando como as placas vermelhas, após a inversão, assemelharam-se às placas cinzas ao ficarem com fundo claro e caracteres escuros e vice-versa.

Figura 4.23 – Exemplos de Placas Utilizadas no Treinamento do Modelo de Segmentação

Fonte: Autor.

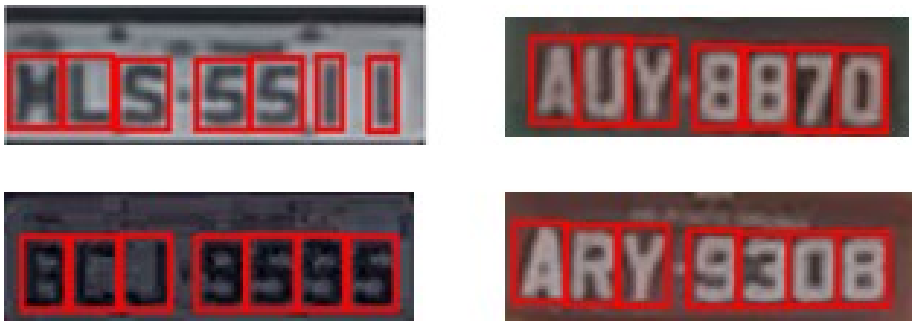
4.2.2 Preparo do Modelo de Segmentação

Após concluir o preparo dos rótulos e imagens, foi realizado o treinamento do YOLOv5m, definido na seção anterior, para poder realizar a segmentação. Para realizar o treinamento, foram utilizados os recursos de computação em nuvem da plataforma Google Colab (39).

Em termos de parâmetros utilizados para o treinamento, foram utilizadas 50 épocas de treinamento com 64 imagens compondo cada *batch*. Desta forma, a cada 64 imagens do conjunto de treinamento, composto por 2880 imagens, os parâmetros foram ajustados, repetindo este procedimento por 50 vezes. Além disso, foram utilizadas 1.440 imagens para a validação, incluindo imagens com cores invertidas, como mencionado anteriormente, e outras 1.440 imagens para o teste do modelo. Como foi realizado um processo de aumento do conjunto de dados, conforme exposto anteriormente, o número de imagens de treinamento e de validação é o dobro do que foi utilizado para o reconhecimento de placas, enquanto o número de imagens da etapa de testes é o mesmo, visto que, para esta etapa, não foram utilizadas imagens invertidas.

Na Figura 4.24 é possível ver alguns dos resultados obtidos pelo modelo de segmentação, tendo sido utilizadas imagens do conjunto de testes, não vistas anteriormente pela rede neural treinada, para obter esta Figura.

Figura 4.24 – Exemplos de Caracteres Segmentados pelo Modelo Treinado



Fonte: Autor.

Em alguns casos, o modelo identificou mais do que sete objetos na imagem, detectando, algumas vezes, o mesmo caractere duas vezes. Assim, utilizou-se somente as sete detecções com maior confiança calculada pelo modelo, para segmentar os caracteres e repassá-los para a etapa de identificação de caracteres.

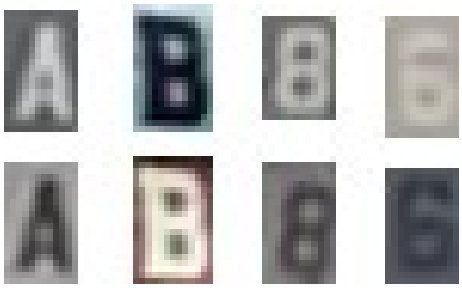
4.2.3 Preparo dos Dados para a Identificação de Caracteres

Para identificar cada um dos caracteres segmentados, conforme mencionado no Capítulo 3, foi definido que seriam utilizados dois modelos de redes neurais convolucionais, um para identificação de letras e outro para identificação de números. Esta abordagem foi utilizada com o objetivo de evitar que o modelo treinado confunda letras com números e vice-versa, tendo obtido bons resultados em outros trabalhos (29). Dessa forma, foi realizado o preparo de duas bases de imagens, uma para letras e outra para números.

Nesse sentido, utilizando os rótulos originais, os caracteres foram recortados e armazenados nestas bases. Para isso, foram adicionados *pixels* pretos para igualar as dimensões de cada imagem e, em seguida, foi feito o redimensionamento das imagens para 30 por 20 *pixels* com o objetivo de padronizar as imagens do conjunto, visto que todos os caracteres precisam ter as mesmas dimensões para serem utilizados com os modelos de reconhecimento de caracteres. Além disso, assim como feito para as placas, foram geradas imagens com cores invertidas para cada caractere recortado, dobrando o conjunto de dados, visando aumentar a diversidade e quantidade de dados para obter redes neurais mais robustas. A Figura 4.25 ilustra

alguns destes caracteres, em que é possível ver sua versão original e sua versão em negativo.

Figura 4.25 – Exemplos de Caracteres do Conjunto de Treinamento



Fonte: Autor.

Para o reconhecimento de caracteres, também foram utilizadas as imagens de letras e números pertencentes a placas de motos, motonetas e veículos similares que não foram consideradas nos passos anteriores. Como não há diferenças significativas entre imagens de caracteres recortados de placas de carros e caracteres recortados de placas de motos, por exemplo, estas imagens foram adicionadas nos conjuntos de letras e números para aumentar a quantidade e diversidade de imagens dos bancos de dados.

Dessa forma, o conjunto de letras foi construído com 10.800 e 5.400 imagens para os conjuntos de treinamento e validação, enquanto o grupo de imagens de números totalizou 14.400 e 7.200 imagens para treinamento e validação, respectivamente.

Em sequência, foi realizado o preparo dos rótulos de cada caractere. Diferentemente das etapas de reconhecimento de placas e segmentação, foi necessário informar apenas a classe do objeto, não sendo necessário informar as coordenadas, visto que o caractere é repassado já recortado para o modelo de reconhecimento.

Após preparar as imagens dos conjuntos de treinamento e validação, foi realizado a normalização dos valores dos *pixels* de cada imagem. Esse processo consistiu em enquadrar os valores dos *pixels,* que variam entre 0 e 255 para uma imagem de 8 *bits*, em um intervalo de 0 a 1, com o objetivo de melhorar os resultados obtidos pela rede treinada, facilitando a convergência do modelo (44).

Além disso, como as saídas dos modelos são vetores binários com tamanhos iguais ao número de classes para que as redes foram treinadas, foi necessário converter os rótulos para o mesmo formato. Por exemplo, para a letra B, que possuí rótulo igual a 1, após passar pela conversão, passa a ser representado pelo vetor [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. Desta forma, o valor 1, na segunda posição do vetor, indica que imagem pertence à segunda classe, ou seja, à classe B, sendo cada posição do vetor, uma classe do alfabeto, e o valor 1 indicando que é verdadeiro que aquele objeto pertence àquela classe.

4.2.4 Treinamento dos Modelos de Reconhecimento de Caracteres

Após a finalização da etapa de preparo das imagens e rótulos, foi realizado o treinamento das duas redes neurais convolucionais mencionadas anteriormente.

Para a rede neural convolucional dedicada para o reconhecimento de letras, foram definidas 12 camadas de convolução, 3 camadas de *pooling* e uma camada de saída contendo 26 neurônios, um para cada letra, utilizando a função de ativação "*softmax*". Além disso, para evitar *overfitting*, conforme apresentado anteriormente, o hiperparâmetro *dropout* foi definido em 25% para a última camada de convolução.

A Tabela 4.1 resume toda a estrutura da CNN construída. A primeira coluna indica a ordem das camadas, enquanto a segunda indica o tipo ou função de cada uma. A terceira coluna, por sua vez, contém o formato dos dados de saída. Por ela, é possível notar a quantidade de filtros que foi utilizado em cada camada convolucional para extrair informações da imagem. Por exemplo, a primeira camada foi definida para utilizar 32 filtros, enquanto a última foi configurada com 155 filtros. Por fim, a última coluna indica quantos parâmetros a camada possui e foram ajustados durante o processo de treinamento. Esse valor muda de acordo com as dimensões definidas para a imagem de entrada, e a quantidade de filtros utilizados por camada convolucional.

A camada "*flatten*", por seu turno, realiza o reordenamento dos valores de saída da camada anterior para que eles possam ser utilizados para excitar os neurônios da camada de saída. Isto é, como a saída de uma camada convolucional é uma matriz, estes valores são transformados em um único vetor unidimensional, por exemplo, para

a última camada, a saída com formato $3 \times 2 \times 155$ foi transformada em um vetor com 930 elementos, como a Tabela 4.1 demonstra.

Finalizada a construção da rede neural responsável pelo reconhecimento de letras, foi feito o preparo da estrutura dedicada aos números. Para isso, a maior parte da estrutura descrita pela Tabela 4.1, foi reaproveitada, alterando-se somente a quantidade de neurônios na camada de saída e a quantidade de filtros na última camada convolucional. Dessa forma, a saída, camada 17 pela Tabela 4.1, foi configurada com 10 neurônios, visto que as imagens dos números podem assumir somente uma dentre 10 classes possíveis. Ademais, foi utilizado somente 75 filtros na camada 15 da Tabela 4.1.

Tabela 4.1 – Estrutura da Rede Neural Convolucional de Identificação de Letras

#	Camada (Tipo)	Formato de Saída	Número de Parâmetros Ajustáveis
1	Convolucional	$30 \times 20 \times 32$	896
2	*MaxPooling*	$15 \times 10 \times 32$	0
3	Convolucional	$15 \times 10 \times 64$	18496
4	*MaxPooling*	$7 \times 5 \times 64$	0
5	Convolucional	$7 \times 5 \times 128$	73856
6	Convolucional	$7 \times 5 \times 64$	8256
7	Convolucional	$7 \times 5 \times 128$	73856
8	*MaxPooling*	$3 \times 2 \times 128$	0
9	Convolucional	$3 \times 2 \times 256$	295168
10	Convolucional	$3 \times 2 \times 128$	32896
11	Convolucional	$3 \times 2 \times 256$	295168
12	Convolucional	$3 \times 2 \times 512$	1180160
13	Convolucional	$3 \times 2 \times 256$	131328
14	Convolucional	$3 \times 2 \times 512$	1180160
15	Convolucional (*Dropout*)	$3 \times 2 \times 155$	79515
16	*Flatten*	930	0
17	*Dense*	26	24206

Fonte: Autor.

Para calcular a quantidade de filtros, foi utilizada a seguinte expressão (29):

$$filters = (C + 5) \cdot A \tag{4.10}$$

Em que a quantidade de filtros, "*filters*", é igual ao produto do número de caixas âncora "A" com o número de classes, "C", que a rede foi projetada para reconhecer, acrescido de 5. Neste caso, como a quantidade de filtros depende da quantidade de classes, isto justifica a menor quantidade de filtros para o modelo de reconhecimento de números na camada 15.

Dessa forma, após a definição da estrutura dos modelos, foram definidas 30 épocas de treinamento para o modelo de reconhecimento de letras, com *batch* de 128 imagens. Já para o modelo de reconhecimento de números, foram definidas apenas 15 épocas, visto que a quantidade de épocas utilizada para o modelo de reconhecimento de letras se mostrou maior do que o necessário.

Após finalizar o treinamento dos modelos de reconhecimento de placas, segmentação e identificação de caracteres, para ser possível utilizar as redes obtidas dentro de *smartphones* Android junto com a biblioteca TensorFlow Lite (37), foi necessário converter os arquivos dos modelos para o formato TFLITE utilizando a função "*export*" do YOLOv5 e a função "*tf.lite.TFLiteConverter.from_keras_model*" da biblioteca keras, que é uma biblioteca de código aberto para *Deep Learning*, implementada para ser usada com várias linguagens, como a linguagem Python (45).

4.3 CONSTRUÇÃO DO APLICATIVO *MOBILE*

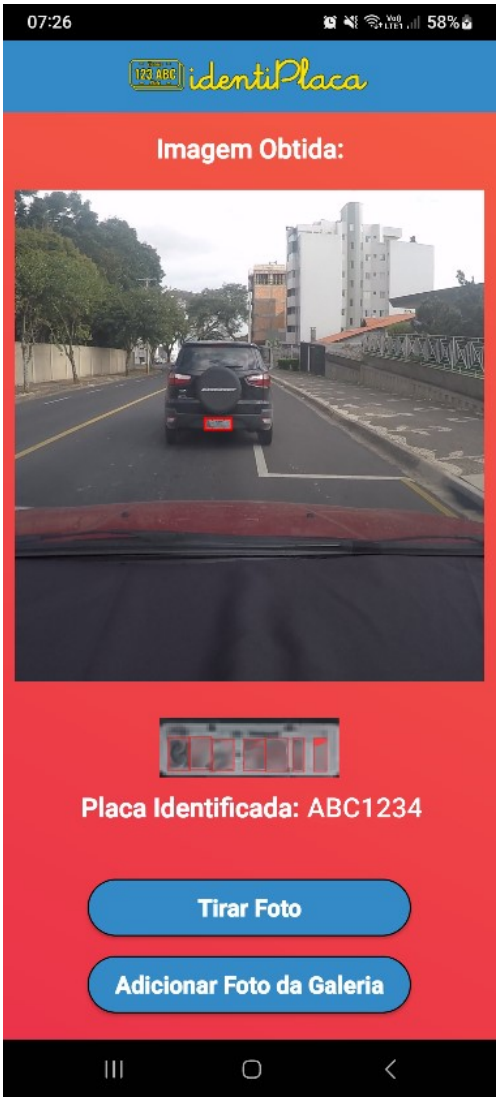
Nesta seção, o desenvolvimento da aplicação *mobile* será apresentado, expondo como foi construída a interface gráfica da aplicação e como os modelos treinados anteriormente são utilizados para a identificação completa dos caracteres das placas.

4.3.1 Interface do Usuário (*Front-End*)

Para a construção da aplicação, foi desenvolvido o *front-end*, ou seja, a interface gráfica que o usuário utiliza para interagir com a aplicação. Para isso, foram inseridos dois botões, um para o usuário acionar a câmera e obter uma imagem, e outro botão para abrir os arquivos do *smartphone*, permitindo que seja utilizada uma imagem previamente salva no dispositivo.

A Figura 4.26, a seguir, mostra o *front-end* construído para a aplicação, utilizando uma imagem de um veículo do conjunto de testes como exemplo. Por questões de privacidade e segurança, os dados da placa foram omitidos.

Figura 4.26 – Captura de Tela da Aplicação Desenvolvida



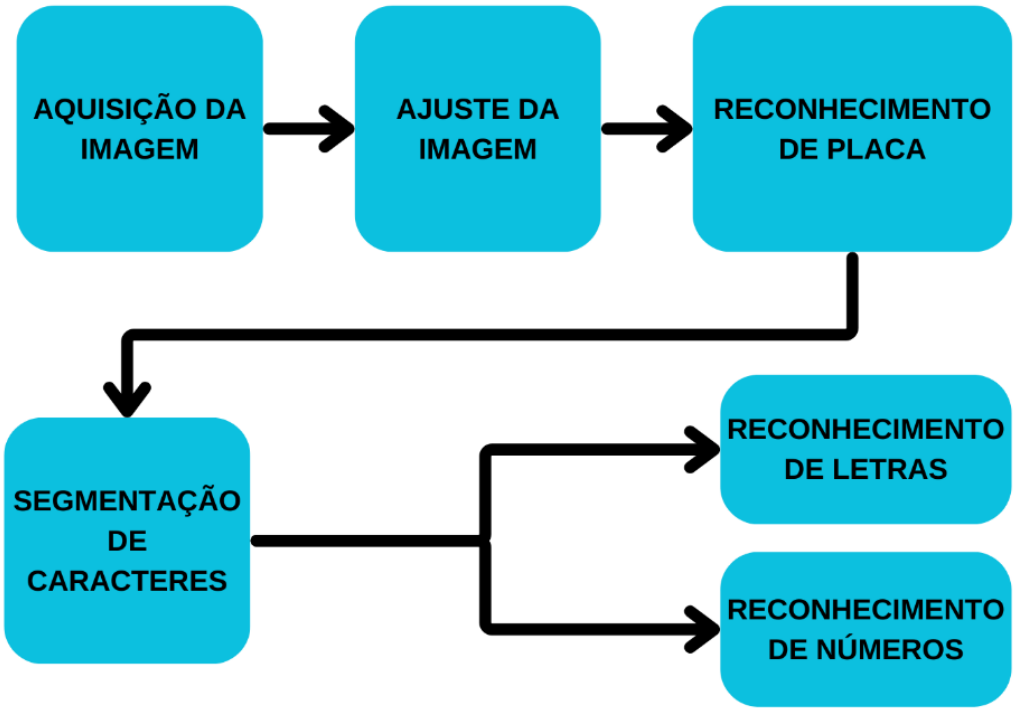
Fonte: Autor.

Como pode ser visto, no topo do *front-end* foi adicionado um contêiner para mostrar a imagem capturada e processada para o usuário, isto é, após realizar o reconhecimento completo da placa de algum veículo, este componente apresenta a imagem com a região da placa delimitada por um retângulo vermelho. Também, foi adicionado um segundo contêiner, abaixo do anterior, com a função de mostrar a placa recortada com os caracteres segmentados. Por fim, uma caixa de texto foi inserida para apresentar os resultados da identificação da placa para o usuário.

4.3.2 Funcionalidades da Aplicação (*Back-End*)

Para a construção do *back-end* da aplicação, ou seja, da parte não visível ao usuário, mas que orquestra o funcionamento do *app*, foi definido o fluxo de ações a serem realizadas, como pode ser visto pela Figura 4.27.

Figura 4.27 – Fluxo de Operação da Aplicação



Fonte: Autor.

Assim, conforme o fluxo, a primeira ação realizada pela a aplicação é realizar a aquisição da imagem, que ocorre quando o usuário escolhe tirar uma foto ou buscar

uma imagem presente na galeria de imagens do *smartphone*. Ao pressionar o botão de tirar foto ou adicionar imagem da galeria, a aplicação verifica as permissões conferidas a ela e, caso existam, o *app* abre a câmera e salva a imagem capturada pelo usuário ou abre a galeria de imagens para que uma foto seja escolhida. Na etapa seguinte, ocorrem ajustes na imagem recebida pela aplicação, como o redimensionamento para 1056 por 1056 *pixels* e correção da orientação. Após isso, a imagem segue para a próxima etapa.

Na etapa de reconhecimento de placa, o algoritmo cria uma instância do modelo de reconhecimento de placas treinado anteriormente. Então, é feita a normalização dos *pixels* da imagem assim como foi feito para treinar os modelos de reconhecimentos de caracteres. Posteriormente, a foto é transformada em um vetor unidimensional e processada pelo modelo.

Após processar a imagem, a saída da rede neural é obtida na forma de um vetor unidimensional, que engloba todas as informações pertinentes a classes, confiança e coordenadas dos objetos detectados na imagem. A maioria das detecções contidas neste vetor costuma possuir o valor de confiança bem baixo. Além disso, muitas são pertinentes ao mesmo objeto, porém, com confiança e coordenadas ligeiramente diferentes uma da outra. Dessa forma, para coletar a posição da placa, foi utilizada a detecção com maior confiança. Então, utilizando as coordenadas obtidas, a placa é recortada da imagem original, redimensionada para 256 por 256 *pixels*, enviada para a próxima etapa e exibida na tela da aplicação. Além disso, a aplicação aproveita as coordenadas obtidas para desenhar um retângulo vermelho em torno da placa detectada na imagem original, conforme pode ser visto na Figura 4.26.

Na etapa seguinte, referente à segmentação dos caracteres presentes na placa veicular, similarmente ao que foi feito para o reconhecimento de placas, a imagem é normalizada e transformada em um vetor unidimensional. Então, após processar a imagem, é obtida a saída do modelo de segmentação. Feito isso, para conseguir as coordenadas dos caracteres, como não basta apenas utilizar as sete predições mais confiáveis, pois algumas delas poderiam ser detecções repetidas do mesmo objeto, são realizados os seguintes passos: 1) Todas as detecções com confiança menor do que 0,6 são descartadas pela aplicação; 2) As coordenadas das detecções restantes são comparadas entre si utilizando a Intersecção sobre a União (IoU); 3) Como detecções com elevada IoU representam o mesmo objeto detectado, somente uma

das detecções é mantida para estes casos, enquanto as outras são descartadas; 4) Por fim, os caracteres detectados são reordenados com base nas coordenadas do eixo horizontal para refletirem a ordem dos caracteres da placa, visto que, em placas veiculares brasileiras, a posição do caractere revela se ele é uma letra ou um número.

Após a segmentação dos caracteres ser finalizada, as coordenadas obtidas são utilizadas tanto para recortar os caracteres e enviar para os modelos de identificação de números e letras, quanto desenhar retângulos vermelhos em torno dos objetos identificados na placa segmentada, como pode ser visto na Figura 4.26. Como, em placas veiculares do modelo antigo, os três primeiros caracteres são sempre letras, enquanto os quatro últimos são sempre números, as imagens dos caracteres recortados foram enviadas para os respectivos modelos de identificação baseado nas coordenadas obtidas.

Nas etapas seguintes, as redes neurais convolucionais de identificação de letras e números são instanciadas, enquanto as imagens são ajustadas para que possam ser processadas pelos modelos. Assim, primeiro as imagens são redimensionadas para 30 por 20 *pixels*, então, os valores dos *pixels* são normalizados e a imagem é transformada em um vetor unidimensional. Após os ajustes, os modelos treinados processam as imagens.

Após a execução das redes neurais convolucionais, são obtidos vetores em que a posição com maior valor em cada vetor revela qual letra ou número foi identificado pelo modelo. Dessa forma, a aplicação verifica quais posições possuem maiores valores, converte para os respectivos caracteres e, então, finaliza o ciclo de identificação dos caracteres de uma placa veicular, apresentando os caracteres da placa identificada no *front-end* da aplicação e salvando a imagem original do veículo junto com o horário em que o processamento foi realizado, e os caracteres que foram identificados na memória do dispositivo móvel. Após finalizar o fluxo, a aplicação continua apresentando os dados da última imagem processada, enquanto aguarda uma nova imagem para executar o mesmo fluxo novamente.

4.4 APLICAÇÃO DO SISTEMA DESENVOLVIDO NA IDENTIFICAÇÃO DE VEÍCULOS FURTADOS

Com o objetivo de demonstrar um caso de uso do sistema desenvolvido, foi projetada uma etapa de consulta da situação veicular, isto é, além de reconhecer os caracteres da placa, o *app* também consulta um sistema *online* para verificar se o veículo pertinente à placa identificada se encontra regular ou furtado.

Para implementar esta etapa, primeiramente, buscou-se acesso à base de dados oficial que a aplicação gratuita, SINESP Cidadão, utiliza. Porém, não foram identificados meios de integrar esta base com aplicações desenvolvidas por terceiros, como é o caso do *app* construído no contexto deste trabalho. Então, uma ferramenta chamada API Placas, desenvolvida pela WD Desenvolvimento (46), foi utilizada para realizar a identificação da situação veicular das placas identificadas pela aplicação.

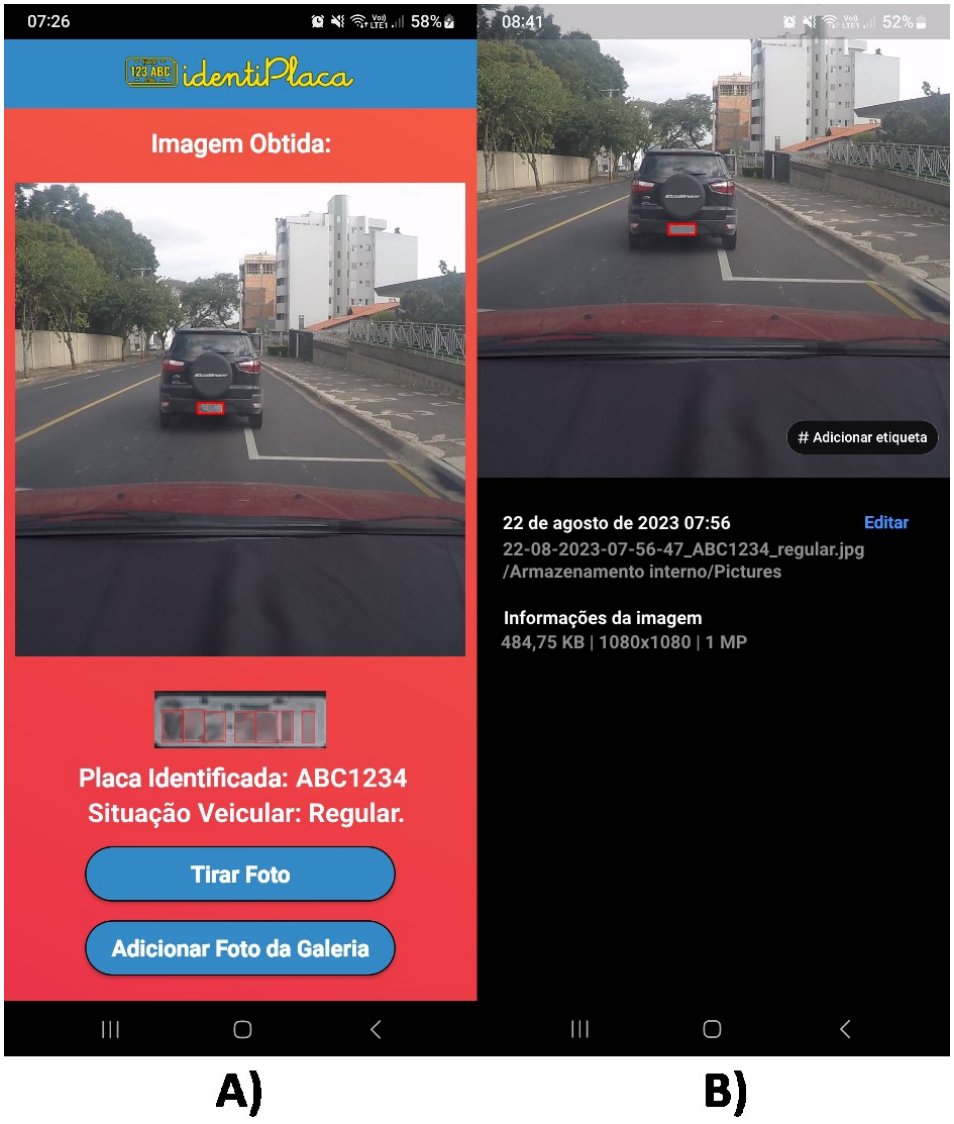
Este recurso permite obter informações do veículo, como a situação veicular, modelo, tabela FIPE, dentre outras, utilizando uma API (*Application Programming Interface*) para habilitar a comunicação entre a aplicação móvel e o servidor do API Placas. Para utilizá-la, foi necessário realizar o cadastro no *website* (46) da ferramenta e contratar o plano de testes, que fornece um *token* que pode ser utilizado para realizar a consulta de dez placas veiculares. Para consultar informações que são alteradas diariamente, como o indicador que revela se o veículo foi roubado ou furtado, é recomendado que um plano *premium* seja contratado. Neste caso, os planos podem ser construídos de acordo com o número de consultas a serem realizadas. Por exemplo, em um orçamento realizado em fevereiro de 2024, o pacote de 10.000 consultas pelo plano *premium* custaria R$ 600,00, ou seja, R$ 0,06 por consulta (46).

Além desta API, outras ferramentas foram encontradas, como a Consulta Online Senatran, que é disponibilizada pelo próprio Governo. Para utilizar este recurso, além de ser necessário pagar pelas consultas, assim como ocorre para o API Placas, ainda é preciso que o usuário tenha seu acesso aprovado pelo Senatran (47). Dessa forma, optou-se por utilizar o API Placas por disponibilizar um pacote de consultas gratuito e não depender de aprovação prévia.

Nesse sentido, para utilizar este recurso, foi utilizado o “Retrofit”, que é uma biblioteca que facilita a realização de requisições utilizando o protocolo HTTP (*Hypertext Transfer Protocol*), habilitando comunicações entre diferentes aplicações

(48). Após realizar a chamada do serviço e obter as informações desejadas no formato JSON (*JavaScript Object Notation*) (46), a aplicação exibe a situação veicular na tela inicial do *app* abaixo da placa identificada. Ademais, a imagem do veículo é salva na memória do dispositivo no formato JPG juntamente com a situação veicular, os caracteres da placa e o horário de processamento da imagem. A seguir, na Figura 4.28, é possível ver um exemplo de caso de uso. Em A), é possível ver a tela da aplicação informando a situação do veículo analisado, bem como em B), há a imagem salva no dispositivo, juntamente com as informações mencionadas anteriormente. Por questões de privacidade e segurança, as informações reais da placa foram omitidas.

Figura 4.28 – Exemplo de Aplicação do Sistema de Reconhecimento de Placas



Fonte: Autor.

4.5 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentadas as diferentes etapas de desenvolvimento que compõem este trabalho, detalhando as tecnologias e plataformas utilizadas, como o YOLO, as Redes Neurais Convolucionais, o Android Studio, dentre outros recursos utilizados. Ademais, um exemplo de caso de uso foi implementado utilizando uma API e os caracteres reconhecidos para identificar a situação da placa veicular processada, ou seja, se ela é relativa a um veículo roubado ou não. Nesse sentido, no próximo capítulo, os resultados obtidos serão apresentados e discutidos.

CAPÍTULO 5

RESULTADOS E DISCUSSÕES

Neste capítulo, serão apresentados e discutidos os resultados do projeto desenvolvido, ressaltando o desempenho individual de cada etapa, bem como o desempenho coletivo da aplicação.

5.1 RESULTADOS DO MODELO DE RECONHECIMENTO DE PLACAS

No intuito de avaliar a performance do modelo de reconhecimento de placas, foi verificado quantas placas do conjunto de testes foram devidamente reconhecidas. Para isso, foi definido que a detecção correta de uma placa veicular envolve delimitar toda a região que contém seus caracteres constituintes, visto que obter apenas parte da placa é insuficiente quando considerado o objetivo de realizar a identificação completa do veículo.

Nesse sentido, para calcular quantas placas foram detectadas corretamente pelo modelo treinado, foi realizado o cálculo da Interseção sobre a União (IoU) entre as coordenadas obtidas pelo modelo e as coordenadas dos rótulos originais. Para detecções com IoU maior ou igual a 70%, a detecção da placa foi considerada como correta. Este valor de limiar foi obtido empiricamente, tendo sido observado que detecções com IoU menor do que isso não englobam totalmente os caracteres, podendo ser casos de falsos positivos ou detecções de apenas uma parte da placa.

Dessa forma, foi obtido 95,35% de taxa de acerto para as imagens do conjunto de testes que possuem pelo menos uma placa veicular totalmente visível, sendo 1.291 imagens nesta condição em um total de 1.440 fotos. Logo, é possível ver que a rede neural treinada conseguiu identificar corretamente a grande maioria das placas do conjunto de testes. Ademais, o resultado está em consonância com a taxa de acerto obtida em outro trabalho que utilizou o YOLO para a mesma tarefa, obtendo sucesso em identificar a placa veicular em 98,33% dos casos (26).

A Figura 5.29 ilustra casos de falsos positivos obtidos pelo modelo treinado. Como é possível ver, alguns objetos foram indevidamente detectados como placas

devido à semelhança de cores e formatos. Quando ocorreram estes casos, foi verificado que a rede neural treinada também reconheceu os objetos que realmente são placas veiculares, porém, como o sistema desenvolvido utiliza somente a detecção com maior confiança calculada pelo modelo e obtida em sua saída, juntamente com as coordenadas do objeto detectado, não foi possível reconhecer os caracteres das placas dos veículos destas imagens, visto que a rede neural atribuiu confianças menores para as detecções das placas veiculares nestes casos específicos.

Figura 5.29 – Exemplo de Objetos Identificados Incorretamente

Fonte: Autor.

Para esta situação, não foi implementada uma solução, porém, para uma futura versão do sistema, uma melhoria que pode ser feita é utilizar não somente o objeto detectado com maior confiança, mas também, as detecções com valores próximos, de forma que, em etapas posteriores, seria verificado qual objeto é mais provável de

ser a placa veicular, utilizando, por exemplo, a quantidade de caracteres segmentados com elevada confiança para cada objeto.

Somando-se aos casos de falsos positivos, as ocorrências de detecções de apenas partes das placas veiculares também contribuíram para que a taxa de acerto não fosse 100%. A Figura 5.30 exemplifica um caso destes, sendo possível ver que uma parte do último caractere da placa ficou parcialmente fora da detecção. O retângulo vermelho representa a detecção feita pelo modelo, enquanto o retângulo azul indica a parte do caractere da placa veicular que ficou de fora da detecção. Como é necessário que todos os caracteres sejam corretamente identificados para identificar uma placa, não é possível atingir este fim para situações como esta, visto que os modelos de segmentação e identificação demandam que os caracteres estejam presentes em sua totalidade para que funcionem adequadamente, segmentando e identificando cada um.

Para estes casos, também não foram implementadas soluções, mas, um possível ajuste que pode ser feito, é analisar a região presente ao redor da placa detectada e aplicar algoritmos de processamento digital de imagens para identificar possíveis bordas da placa que possam ter ficado de fora da detecção.

Figura 5.30 – Exemplo de Placa Parcialmente Detectada

Fonte: Autor.

Dentre outros parâmetros utilizados para avaliar o modelo de reconhecimento de placas, foram calculadas as métricas mAP@50 e mAP@50-95. Estendendo os conceitos vistos na seção 2.5.7, o mAP@50 pode ser descrito como o cálculo do mAP considerando limiar de 0,5 para a Intersecção sobre a União (IoU) (49), ou seja, é o cálculo do mAP considerando como detecções corretas somente aquelas que se sobreponham 50% ou mais com o objeto real. Dessa forma, retomando a equação (2.8), para o mAP@50, n assume o valor 1, visto que há somente um único valor de limiar. Além disso, como a rede neural treinada para reconhecer placas veiculares trabalha com somente uma classe, na Equação (2.9), n_c também assume o valor 1 e, portanto, para modelos treinados para reconhecer somente uma classe de objeto, como é o caso deste, o mAP@50 é igual ao produto do *Recall* pela precisão assumindo IoU como 0,5.

O mAP@50-95, por sua vez, é o cálculo do mAP para valores de IoU variando entre 0,5 e 0,95. Como o modelo de reconhecimento de placas veiculares detecta somente uma classe de objeto, o mAP@50-95, para este caso, é igual ao AP@[.5:.95], apresentado na Seção 2.5.7. Como essa métrica considera valores maiores de Intersecção sobre a União para o cálculo, seu valor tende a ser menor do que o mAP@50, visto que a obtenção de detecções com IoU mais próximo de 1 é mais difícil de acontecer devido à necessidade de que o modelo obtenha em suas predições, coordenadas extremamente fiéis às coordenadas reais dos objetos detectados. Por conta disso, o mAP@50-95 é muito útil para verificar a qualidade do modelo treinado para tarefas que requerem detecções precisas (49). Assim, valores elevados para o mAP@50-95 indicam que a rede neural treinada está conseguindo detectar os objetos e, além disso, está obtendo detecções com coordenadas muito próximas das coordenadas reais, enquanto valores elevados para o mAP@50 indicam que o modelo está detectando os objetos, embora, não necessariamente, as detecções estejam com coordenadas muito precisas. Nesse sentido, ambas as métricas são cruciais para avaliar como está a performance de um modelo de detecções de objetos (49).

Após realizar o cálculo destas métricas, para o mAP@50, foi obtido 0,974 enquanto o mAP@50-95 resultou em 0,779. Dessa forma, considerando o que foi exposto anteriormente, é possível afirmar que a rede neural identifica a região contendo a placa veicular muito bem, porém, as coordenadas obtidas pelo modelo divergem um pouco das coordenadas reais, o que é sintetizado pelo valor do mAP@50-95, que é ligeiramente menor do que o mAP@50. Considerando que a

etapa posterior não depende de detecções com coordenadas extremamente fiéis às coordenadas reais da placa veicular para conseguir segmentar os caracteres, bastando todos os caracteres estarem totalmente presentes na imagem da placa obtida na detecção, conclui-se que o modelo de reconhecimento de placas possui desempenho bastante satisfatório.

5.2 PERFORMANCE DO MODELO DE SEGMENTAÇÃO DE CARACTERES

Para obter a performance do modelo de segmentação de caracteres, obtido na fase de desenvolvimento, foi realizada uma abordagem similar à utilizada na seção anterior. A principal diferença entre as metodologias empregadas é que o modelo de segmentação precisa identificar corretamente os sete caracteres que compõem cada placa veicular para que a segmentação de uma placa seja considerada como correta. Este critério foi definido pois, caso o caractere não seja segmentado ou seja parcialmente segmentado, ele não é recortado e enviado para a etapa de reconhecimento de caracteres. Consequentemente, a identificação completa da placa veicular, que é o objetivo principal do sistema desenvolvido, falha como um todo.

Nesse sentido, foi estabelecido empiricamente que a Intersecção sobre a União de cada predição com o objeto real deve ser maior ou igual a 35% para garantir a segmentação correta do caractere. Este valor de limiar foi obtido após testes com diferentes valores, tendo sido verificado que, para alguns caracteres, como a letra “I” e o número “1”, mesmo após segmentar o caractere corretamente, pequenas diferenças entre as coordenadas da predição e do objeto real provocaram valores baixos para a Intersecção sobre a União, o que motivou utilizar um valor de limiar mais baixo, ou seja, de 35% para considerar uma segmentação como correta. Por consequência, houve a necessidade de verificar se este valor de limiar não provocava a contabilização de segmentações parciais de caracteres como detecções corretas. Para isso, todas as imagens com IoU maior do que 0,35 e menor do que 0,35 foram analisadas, tendo sido observado que o valor de limiar definido empiricamente não causou o referido problema, ou seja, não contabilizou detecções de apenas partes de caracteres como corretas, e, portanto, pode ser utilizado para identificar segmentações corretas e incorretas.

Dessa forma, foi obtido 99,86% de taxa de acerto para as placas do conjunto de testes. Assim, considerando que o total de imagens de placas veiculares utilizadas foi 1.440, em apenas duas, a segmentação falhou em obter os sete caracteres corretamente. Após investigar as detecções realizadas nestas duas imagens, foi verificado que o modelo não detectou o último caractere em ambos os casos. Para estas situações, não foram tomadas medidas, visto que os resultados obtidos para a segmentação foram satisfatórios, no entanto, para cenários em que latência e capacidade computacional para processar imagens não são problemas, uma solução híbrida utilizando processamento digital de imagens poderia ser utilizada para segmentar o caractere restante. Por exemplo, como a distância entre os caracteres tende a ser a mesma, pelas posições dos objetos já identificados pela rede neural treinada, a provável região contendo o caractere faltante poderia ser processada para delimitar a letra ou número faltante.

Ademais, após calcular a performance da rede utilizando outras métricas, foi verificado que, para o parâmetro *Mean Average Precision* (mAP) ou mAP@50, a rede treinada obteve 0,995, enquanto para o mAP@50-95, o modelo pontuou 0,691, tendo como base de cálculo, as detecções realizadas para o conjunto de imagens de testes. Portanto, é possível afirmar que o modelo do YOLO treinado para realizar a segmentação possui excelente performance para detectar os caracteres, embora as predições não tenham elevada Intersecção sobre a União com as coordenadas reais dos caracteres. No entanto, como apresentado anteriormente, mesmo a IoU não sendo elevada, não ocorreram casos de caracteres parcialmente detectados.

Em termos comparativos, a rede neural obtida possui desempenho similar a outros modelos construídos em outros trabalhos, como a CR-NET, que é uma solução baseada, assim como o YOLO, em Redes Neurais Convolucionais, preparada para segmentação e identificação de caracteres, que atingiu 95,97% de taxa de acerto para a mesma tarefa (29).

5.3 DESEMPENHO DO RECONHECIMENTO DE CARACTERES

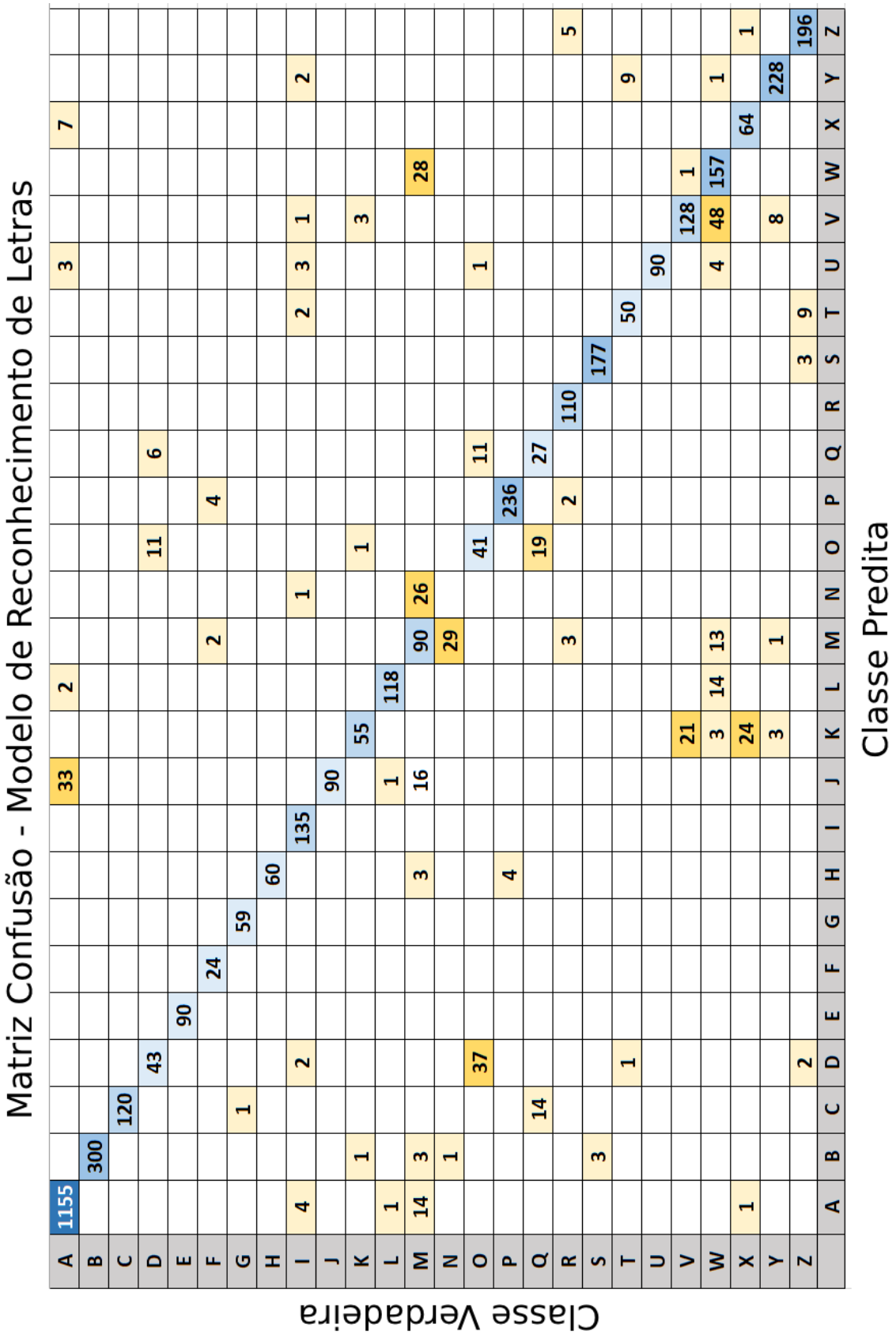
Após realizar a avaliação dos modelos de reconhecimento de placas e segmentação de caracteres, foi realizado a análise de performance das duas redes neurais convolucionais construídas de acordo com as estruturas apresentadas na seção 4.2.4. Para isso, os caracteres de cada placa veicular do conjunto de testes foram recortados utilizando os rótulos originais, e foram processados pelos respectivos modelos de identificação de letras e números. Para calcular a taxa de acertos, foi considerado que uma placa veicular foi devidamente reconhecida somente para casos em que os modelos conseguiram identificar corretamente os sete caracteres constituintes.

Dessa forma, baseado nesse critério, a taxa de acerto obtida foi de 66,18%, ou seja, em um universo de 1.440 imagens de placas de veículos, como carros, caminhões, ônibus, dentro outros, excluindo-se motos, motonetas e outros veículos contendo placas com duas linhas de caracteres, os modelos de identificação de letras e números conseguiram acertar os sete caracteres de cada placa em 953 das imagens.

Em termos de performance individual, foi verificado que a rede neural convolucional responsável por identificar letras, conseguiu identificar as três letras constituintes de cada placa em 73,06% dos casos, enquanto o modelo de reconhecimento de números identificou os quatro algarismos em 84,93% das imagens. Como basta um caractere errado para invalidar o reconhecimento da placa, o resultado de 66,18% está dentro do esperado, visto que, em alguns casos, os modelos acertaram todos os números, mas falharam em uma letra, e vice-versa.

Com o intuito de analisar a performance de cada rede neural treinada para cada classe, foi obtido a Matriz Confusão referente a cada modelo. Conforme apresentado na seção 2.5.1, a Matriz Confusão relaciona as detecções realizadas com as classes verdadeiras de cada imagem, com isto, é possível, por exemplo, analisar se a rede neural obtêm melhores resultados para uma classe do que para outra. Nesse sentido, a Figura 5.31 apresenta a Matriz Confusão para o modelo dedicado ao reconhecimento de letras.

Figura 5.31 – Matriz Confusão Referente ao Modelo de Identificação de Letras



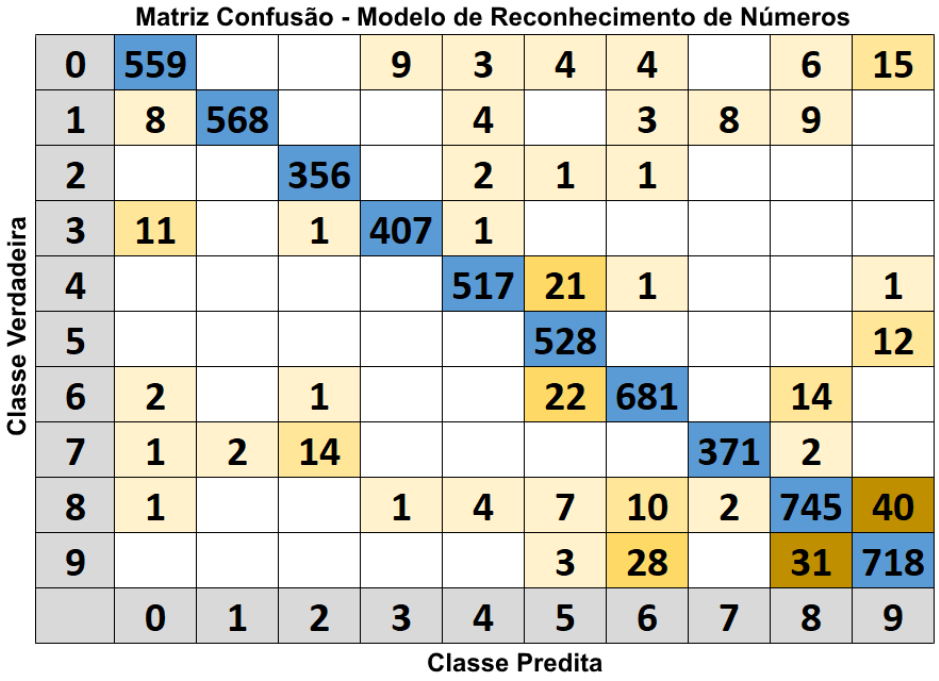
Fonte: Autor.

Dessa forma, a Matriz Confusão obtida mostra que, para a maioria das letras, o modelo treinado consegue reconhecer o caractere corretamente. Por exemplo, para a letra “A”, a predição do modelo acertou o caractere em 1.155 casos, errando a classificação em 20 ocorrências deste caractere. No entanto, para outras letras, a detecção falha consideravelmente, como é o caso da letra “N”. Neste caso, a rede não acertou a classificação em nenhuma imagem. Em 29 casos, a letra “N” foi classificada como “M”, enquanto em uma imagem foi reconhecida como sendo a letra “B”.

Após investigar os motivos para que as detecções tenham obtido performances significativamente diferentes para alguns caracteres, foi constatado que o número de ocorrências de cada letra no banco de imagens utilizado é razoavelmente diferente. Isso ocorreu, pois, como destacado na Seção 2.4, alguns caracteres são mais frequentes do que outros dependendo de qual Estado brasileiro o veículo foi originalmente emplacado. Como todas as imagens do conjunto utilizado foram obtidas no Paraná, há uma prevalência de alguns caracteres sobre outros. Assim, durante o treinamento do modelo, não deve ter sido possível extrair informações suficientes para que a rede consiga diferenciar adequadamente o caractere “N” de outras letras, enviesando as detecções realizadas para o conjunto de testes. Para estes casos, não foram implementadas soluções, mas possíveis estratégias incluem diversificar a base de dados utilizada, com imagens de veículos de outros Estados, e aplicar outras técnicas de “*data augmentation*” ou aumento de dados, em tradução livre, para gerar imagens artificiais dos caracteres menos recorrentes e utilizá-las no treinamento do modelo.

De maneira análoga, obteve-se a Matriz Confusão para a rede neural responsável por identificar números, como pode ser visto na Figura 5.32.

Figura 5.32 – Matriz Confusão Referente ao Modelo de Identificação de Números



Fonte: Autor.

Para este caso, foi observado que todos os números foram reconhecidos na maioria dos casos, sem grande diferença de performance entre um número e outro, o que era esperado, visto que a quantidade de ocorrências de cada caractere é similar entre cada um, o que dificulta a formação de vieses durante o treinamento do modelo. Somando-se a isto, há a menor quantidade de classes a serem reconhecidas pela rede treinada, o que facilitou a extração de informações suficientes das imagens para que o modelo conseguisse diferenciar melhor cada algarismo um do outro. Ademais, as imagens das placas veiculares utilizadas possuem quatro números e três letras por placa, ou seja, existem mais imagens de caracteres para treinar o modelo de reconhecimento de números do que para treinar a rede responsável por letras. Consequentemente, o modelo de identificação de números tende a ser mais robusto.

Para tratar os casos em que o reconhecimento dos sete caracteres falhou, não foram implementadas outras soluções, conforme mencionado anteriormente, mas uma outra possível abordagem que poderia ser empregada, seria utilizar mais do que uma imagem de um mesmo veículo para determinar sua placa, isto é, poderiam ser obtidas várias imagens em momentos diferentes para o mesmo veículo, identificando os caracteres da placa em cada foto e, após analisar quais caracteres são mais

recorrentes para cada letra e número da placa veicular, utilizá-los para determinar a identificação do veículo.

Este processo foi utilizado em outros trabalhos, tendo contribuído para uma melhora significativa da taxa de acertos, (29). Para ilustrar este processo, a Figura 5.33 mostra a placa identificada para fotos de diferentes momentos do mesmo veículo. A placa veicular correta é dada pelos caracteres AZZ-6958, porém, em uma das detecções, foi obtido ASZ-6958. Isto pode ter ocorrido devido a diversos fatores, por exemplo, o veículo pode ter acelerado no momento da foto, desfocando a região da placa veicular, assim como ocorreu na Figura 2.12. Neste caso, poderiam ser utilizadas as detecções anteriores e subsequentes para identificar que o caractere “Z” é mais recorrente para a segunda posição da placa, corrigindo a detecção da placa. Como, em uma situação real, é provável que um mesmo veículo persista em frente à câmera do dispositivo móvel por tempo suficiente para obter mais de uma foto, utilizar mais do que uma foto do mesmo veículo é uma estratégia bastante viável.

Figura 5.33 – Exemplos de Identificações para a mesma Placa Veicular

79	AZZ	6958	AZZ-6958
80	AZZ	6958	AZZ-6958
81	AZZ	6958	AZZ-6958
82	AZZ	6958	AZZ-6958
83	AZZ	6958	AZZ-6958
84	AZZ	6958	AZZ-6958
85	ASZ	6958	ASZ-6958
86	AZZ	6958	AZZ-6958
87	AZZ	6958	AZZ-6958
88	AZZ	6958	AZZ-6958
89	AZZ	6958	AZZ-6958

Fonte: Autor.

5.4 PERFORMANCE GERAL DO SISTEMA

Além de calcular a performance individual para cada modelo treinado, foi realizada a análise da performance geral do sistema. Para isso, foi verificado para

quantas placas veiculares do conjunto de testes, o sistema conseguiu identificar todos os caracteres corretamente, utilizando a saída de cada modelo como entrada para o próximo. Ou seja, ao invés de utilizar as informações dos rótulos originais de cada placa para avaliar cada modelo separadamente, como feito anteriormente, foram utilizadas as informações das coordenadas obtidas pelas predições realizadas por cada rede neural treinada anteriormente, simulando o que ocorre em um cenário real de uso da aplicação. Por fim, foi analisada a performance de execução da aplicação construída para s*martphones* Android, com o intuito de averiguar a viabilidade de execução do sistema construído dentro de um dispositivo de limitada capacidade computacional.

Dessa forma, primeiramente, as imagens do conjunto de testes foram processadas pelo modelo de reconhecimento de placas. Então, as placas resultantes foram enviadas para a rede neural responsável por realizar a segmentação. Feito isso, os caracteres segmentados seguiram para os modelos de identificação. Considerando que o conjunto de testes possui 1.440 imagens de veículos, com 1.291 fotos em que a placa veicular está totalmente visível, após realizar a análise de performance, foi constatado que o sistema conseguiu identificar corretamente 913 placas, isto é, 70,70% das imagens contendo placas veiculares visíveis tiveram os sete caracteres reconhecidos corretamente. Em termos comparativos, em outro trabalho, os autores obtiveram 64,89% de taxa de acerto, sendo que o conjunto de testes utilizado por eles era composto, também, por placas veiculares com duas linhas de caracteres, como placas de motos (29).

Após obter a taxa de acertos que representa o sistema como um todo, foram coletadas métricas de desempenho computacional da aplicação. Para isso, foram obtidos registros do tempo gasto para executar cada modelo em um cenário real de execução do *app*. Também, foi cronometrado o tempo total gasto para executar a aplicação para uma imagem. Para esta tarefa, foi utilizado um *Smartphone* Samsung A32, que é um aparelho de capacidade computacional intermediária que conta com 4 GB de memória RAM, GPU Mali-G52 MC2, *Chipset* Helio G80 MediaTek e 8 núcleos de processamento divididos em 2 com 2 GHz de *Clock* (Cortex-A75) e 6 com 1,8 GHz (Cortex-A55) (50). Estes testes foram conduzidos para cinco imagens diferentes, tendo sido obtido a média de tempo gasto para processar cada uma. Os resultados podem ser vistos na Tabela 5.2, apresentada a seguir.

Tabela 5.2 – Resultados Obtidos para a Aplicação ALPR Desenvolvida

Etapa	Taxa de Acertos	Tempo Gasto (s)	FPS
Reconhecimento de Placa Veicular	95,35%	1,8	0,56
Segmentação de Caracteres	99,86%	0,86	1,16
Identificação de Caracteres	66,18%	0,20	5,00
Sistema Geral	70,70%	3,0	0,33

Fonte: Autor.

De acordo com os resultados obtidos, o sistema desenvolvido consegue processar uma imagem a cada três segundos, aproximadamente, o que é um intervalo de tempo muito bom para s*martphones* de entrada ou intermediários, considerando que são dispositivos mais limitados em termos de recursos computacionais e que a aplicação precisa executar quatro redes neurais artificiais para realizar a análise de uma imagem contendo uma placa veicular, além de precisar ajustar as imagens, corrigindo as dimensões delas antes de processá-las utilizando cada modelo treinado. Além disso, como era de se esperar, a etapa de reconhecimento de placas gasta mais tempo do que as outras, visto que trabalha com imagens de entrada maiores e que, portanto, demandam mais processamento.

Ademais, a taxa de acerto de 70,70% demonstra que os modelos treinados conseguem reconhecer sete a cada dez placas veiculares, sendo que ainda há espaço para melhorar esta performance utilizando outras técnicas, como mencionado anteriormente. Dessa forma, conclui-se que o sistema é viável para ser utilizado em aplicações como a exemplificada na Seção 4.4, para a detecção de veículos furtados, visto que ela combina uma taxa de acertos razoável com uma boa performance computacional para dispositivos móveis de entrada ou intermediários, levando apenas alguns segundos para obter os caracteres de uma placa veicular presente em uma imagem de um veículo.

CAPÍTULO 6
CONCLUSÃO

Neste capítulo, as considerações finais sobre o trabalho desenvolvido são apresentadas, destacando o que foi feito e obtido, bem como o que pode ser melhorado e construído em trabalhos futuros para enriquecer este projeto.

6.1 CONSIDERAÇÕES FINAIS

Conforme apresentado neste trabalho, tecnologias como o YOLO e as Redes Neurais Convolucionais, têm sido empregadas para a construção de sistemas inteligentes, tornando-se ferramentas que profissionais e estudiosos de diferentes áreas têm buscado expandir seus conhecimentos para utilizá-las em soluções para diferentes problemas.

Nos dias atuais, é possível perceber que há uma grande gama de problemas que podem ser tratados utilizando estas tecnologias, mas ainda não são. O caso de veículos furtados ou roubados é um exemplo. Como destacado anteriormente, hoje, diversas cidades contam com monitoramento realizado por sistemas de câmeras, sendo que estas imagens poderiam ser utilizadas com algoritmos inteligentes para consultar os dados de cada veículo na base de dados utilizada pelo SINESP Cidadão, com o objetivo de identificar veículos irregulares para que as autoridades locais possam atuar. Além disso, com a popularização dos *smartphones*, boa parte dos cidadãos brasileiros contam com dispositivos capazes de obter fotos de veículos e processá-las utilizando modelos de inteligência artificial. Assim, cada indivíduo poderia contribuir com a segurança pública, identificando a situação veicular de automóveis em via pública em situações corriqueiras, notificando as autoridades responsáveis quando a presença de veículos irregulares for identificada por seus aparelhos bastando, para isso, apontar a câmera do *smartphone* para um veículo.

Dessa forma, ao longo deste trabalho, foi construído um sistema para *smartphones* Android capaz de receber imagens contendo veículos, identificar regiões contendo placas veiculares, segmentar e reconhecer os caracteres de cada uma, desde que sejam placas de veículos como carros, ônibus, caminhões, dentre outros tipos de automóveis que possuem placas com somente uma linha de caracteres. Para

construir esta aplicação foi feito uso do algoritmo YOLO para realizar a detecção da placa e a segmentação dos caracteres, e de Redes Neurais Convolucionais para identificar as letras e os números, visto que ambos os algoritmos são conhecidos por obterem bons resultados para tarefas de detecção de objetos sem demandar elevado gasto de recursos computacionais quando comparados com outras tecnologias, o que faz do YOLO e das Redes Neurais Convolucionais excelentes soluções para executar detecção e identificação de objetos em situações de recursos computacionais escassos. Para treinar o YOLO e as Redes Neurais Convolucionais, foram utilizados recursos de computação em nuvem gratuitos e um banco de imagens de veículos brasileiros cedido pela Universidade Federal do Paraná (UFPR).

Considerando que o banco de imagens utilizado conta somente com placas veiculares do modelo antigo, apresentado na seção 2.4, é importante salientar que o sistema desenvolvido não trabalha com as placas do modelo padrão Mercosul, visto que elas possuem uma letra no lugar de um dos números da placa do modelo antigo, o que inviabiliza a lógica utilizada na aplicação construída. No entanto, como as placas do modelo anterior estão sendo gradativamente substituídas pelas placas padrão Mercosul, futuramente será possível considerar que, para todas as placas cujos caracteres foram devidamente segmentados, o quinto caractere mais à direita sempre será uma letra e, portanto, ao invés de utilizar um modelo de identificação de números, a aplicação poderá tratar como um caso de identificação de letras.

Ademais, no decorrer do projeto, com o intuito de demonstrar como pessoas comuns poderiam utilizar esta aplicação no dia-a-dia para contribuir com a segurança pública, foi desenvolvida uma aplicação móvel, que utiliza uma API para descobrir a situação veicular pertinente à placa identificada, revelando se o veículo em questão está regular ou se encontra em situação de furto ou roubo.

De acordo com os resultados individuais obtidos por cada modelo treinado, como a Tabela 5.2, apresentada anteriormente, demonstra, foi verificado que os modelos de localização de placa veicular e segmentação de caracteres, construídos tendo o YOLO como base, atingiram desempenhos excelentes. Em ambos os casos, as redes treinadas conseguiram acurácia superior a 95% considerando as imagens do conjunto de testes. Por sua vez, os modelos de identificação de caracteres obtiveram um desempenho mais modesto, mas com amplo espaço para melhorias, como a utilização de uma base de dados mais diversificada para que as redes treinadas consigam diferenciar melhor um caractere do outro.

Além disso, o resultado obtido avaliando o sistema como um todo também foi satisfatório, reconhecendo corretamente todos os caracteres das placas veiculares em 70,70% das imagens do conjunto de testes. No entanto, é importante ressaltar que os modelos foram treinados, validados e testados utilizando imagens da base de dados UFPR-ALPR. Dessa forma, caso sejam utilizadas imagens de outros conjuntos, os resultado obtidos podem ser prejudicados. Por exemplo, para imagens de veículos emplacados em outras unidades federativas do Brasil, alguns caracteres, menos frequentes na base de dados utilizada neste trabalho, como a letra “N”, podem provocar diminuição na taxa de sucesso de reconhecimento de placas veiculares devido aos eventuais vieses que os modelos de identificação de caracteres devem possuir devido ao fato de que a base de dados utilizada é desbalanceada.

6.2 TRABALHOS FUTUROS

Com o intuito de aprimorar o sistema desenvolvido, como a primeira versão da aplicação faz o reconhecimento de placas veiculares somente para placas com uma linha de caracteres, uma melhoria que poderia ser feita é expandir a capacidade do sistema para conseguir segmentar e reconhecer placas de automóveis com duas linhas de caracteres, como é o caso de motos e outros tipos de veículos. Somando-se a isso, um sistema de votação poderia ser implementado para aumentar a acurácia dos resultados, dessa forma, ao coletar e processar diversas imagens que sejam do mesmo veículo, ao invés de tratar como casos diferentes, o sistema poderia realizar uma verificação, identificando quais caracteres são mais frequentes em cada posição da placa para, então, determinar quais são seus caracteres.

Também, uma base de imagens contendo placas de veículos emplacados em outros Estados brasileiros, exceto o Paraná, poderia ser construída para testar a aplicação desenvolvida, verificando o quanto os resultados podem ser afetados quando a aplicação for apresentada a placas com caracteres menos recorrentes em placas veiculares originárias do Paraná. Ademais, esta base poderia ser construída utilizando câmeras diferentes daquelas utilizadas para construir a base de imagens UFPR-ALPR, visando testar como a aplicação se comporta quando imagens de qualidades diferentes são apresentadas. Somando-se a isso, a inclusão de imagens de veículos de outros Estados brasileiros aos conjuntos de treinamento e validação,

utilizados para treinar e validar os modelos, poderia auxiliar o sistema desenvolvido a obter melhores resultados, minimizando a formação de vieses devido a dados desbalanceados na base de dados utilizada.

Por fim, considerando que, nos dias atuais, coexistem dois padrões de placas veiculares diferentes, um sistema de identificação de tipo de padrão de placa veicular poderia ser desenvolvido, habilitando a identificação de placas do padrão Mercosul.

REFERÊNCIAS

1 MCCARTHY, J. **WHAT IS ARTIFICIAL INTELLIGENCE?**, 2007. Disponível em: http://jmc.stanford.edu/articles/whatisai/whatisai.pdf. Acesso em: 22 maio 2022.

2 RUSSEL, S.; NORVIG, P. **Artificial Intelligence: a modern approach**. 4. ed. Pearson, 2020.

3 IBM. **What is Computer Vision?**, 2019. Disponível em: https://www.ibm.com/topics/computer-vision. Acesso em: 30 maio 2022.

4 ZHU, Y.; HUANG, M.; CHEN, F. **Creating a Real-Time License Plate Detection and Recognition App**, 2021. Disponível em: https://developer.nvidia.com/blog/creating-a-real-time-license-plate-detection-and-recognition-app/. Acesso em: 30 maio 2022.

5 JORNAL EMPRESAS & NEGÓCIOS. **Reconhecimento facial na segurança pública**, 2021. Disponível em: https://jornalempresasenegocios.com.br/tecnologia/reconhecimento-facial-na-seguranca-publica-2/. Acesso em: 03 janeiro 2024.

6 JAVAID, S. **Top 7 Computer Vision Use Cases in Healthcare in 2024**, 2023. Disponível em: https://research.aimultiple.com/computer-vision-healthcare/. Acesso em: 03 janeiro 2024.

7 SACHETO, C. **Roubo de veículos ultrapassa marca de 1 milhão no Brasil em 4 anos**, 2019. Disponível em: https://noticias.r7.com/sao-paulo/roubo-de-veiculos-ultrapassa-marca-de-1-milhao-no-brasil-em-4-anos-11102019. Acesso em: 23 maio 2022.

8 BRASIL. Ministério da Justiça e Segurança Pública. **Sinesp Cidadão**, 2018. Disponível em: https://www.gov.br/mj/pt-br/assuntos/sua-seguranca/seguranca-publica/sinesp-1/sinesp-Cidadao. Acesso em: 23 maio 2022.

9 CÂMARA MUNICIPAL DE PATOS DE MINAS. **Projeto “Olho Vivo” é inaugurado em Patos de Minas**, 2014. Disponível em: https://www.camarapatos.mg.gov.br/index.php/post-formats/noticias/530-projeto-olho-vivo-e-inaugurado-em-patos-de-minas. Acesso em: 04 janeiro 2024.

10 JACKSON, A. et al. Large pose 3D face reconstruction from a single image via direct volumetric CNN regression. In: **Proceedings of the IEEE international conference on computer vision**. 2017. p. 1031-1039. Disponível em: https://openaccess.thecvf.com/content_ICCV_2017/papers/Jackson_Large_Pose_3D_ICCV_2017_paper.pdf. Acesso em: 05 jun. 2022.

11 LARSSON, S.; MELLQVIST, F. **Automatic Number Plate Recognition for Android**, 2019. Dissertação (Bacharelado em Ciências da Computação) – Karlstad University, Karlstad, 2019. Disponível em: https://www.diva-portal.org/smash/get/diva2:1325108/FULLTEXT01.pdf. Acesso em: 12 jun. 2022.

12 SILVA, I. et al. **Artificial Neural Networks: A Practical Course**. Suíça: Springer International Publishing, 2017.

13 MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, v. 5, n. 4, p. 115–133, dez. 1943. Disponível em: https://home.csulb.edu/~cwallis/382/readings/482/mccolloch.logical.calculus.ideas.1943.pdf. Acesso em: 16 abr. 2024.

14 ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6. p. 386-408, 1958. Disponível em: https://doi.org/10.1037/h0042519. Acesso em: 17 abr. 2024.

15 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, out. 1986. Disponível em: https://doi.org/10.1038/323533a0. Acesso em: 17 abr. 2024.

16 RADHAKRISHNAN, P. **What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?**, 2017. Disponível em: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a. Acesso em: 07 jan. 2024.

17 RAKHECHA, A. **Understanding Learning Rate**, 2019. Disponível em: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de. Acesso em: 08 jan. 2024.

18 LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. Disponível em: https://doi.org/10.1109/5.726791. Acesso em: 18 abr. 2024.

19 ALVES, G. **Entendendo Redes Convolucionais (CNNs)**, 2018. Disponível em: https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184. Acesso em: 9 jun. 2022.

20 REDMON, J. et al. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. 2016. p. 779-788. Disponível em: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf. Acesso em: 6 jun. 2022.

21 ALVES, G. **Detecção de Objetos com YOLO – Uma abordagem moderna**, 2020. Disponível em: https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/. Acesso em: 8 jul. 2022.

22 CHRISTIANSEN, A. **Anchor Boxes — The key to quality object detection**, 2018. Disponível em: https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9. Acesso em: 9 jul. 2022.

23 SHARMA, R.; KAUSHIK, B.; GONDHI, N. Character recognition using machine learning and deep learning – a survey. In: **2020 International Conference on Emerging Smart Computing and Informatics (ESCI)**. IEEE, 2020. p. 341-345. Disponível em: https://ieeexplore.ieee.org/document/9167649. Acesso em: 11 jul. 2022.

24 LIU, P.; LI, G.; TU, D. Low-quality license plate character recognition based on CNN. In: **2015 8th International Symposium on Computational Intelligence and Design (ISCID)**. IEEE, 2015. p. 53-58. Disponível em: https://ieeexplore.ieee.org/abstract/document/7469079. Acesso em: 19 jun. 2022.

25 WANG, C.; SU, C. Fast license plate location and recognition using wavelet transform in android. In: **2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA)**. IEEE, 2012. p. 1035-1038. Disponível em: https://ieeexplore.ieee.org/abstract/document/6360875. Acesso em: 19 jun. 2022.

26 CHEN, J. Chinese license plate identification based on Android platform. In: **2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)**. IEEE, 2017. p. 1-5. Disponível em: https://ieeexplore.ieee.org/abstract/document/7977286. Acesso em: 19 jun. 2022.

27 LAROCA, R. et al. On the cross-dataset generalization in license plate recognition. **arXiv preprint arXiv:2201.00267**, 2022. Disponível em: https://arxiv.org/abs/2201.00267. Acesso em: 26 jun. 2022.

28 UOL. **Placa Mercosul: tudo sobre o novo formato de identificação veicular**, 2020. Disponível em: https://www.uol.com.br/carros/noticias/redacao/2020/01/27/placa-mercosul-saiba-o-que-muda-com-o-modelo-adotado-em-todo-o-brasil.htm. Acesso em: 15 jul. 2022.

29 LAROCA, R. et al. A robust real-time automatic license plate recognition based on the YOLO detector. In: **2018 international joint conference on neural networks (ijcnn)**. IEEE, 2018. p. 1-10. Disponível em: https://ieeexplore.ieee.org/abstract/document/8489629. Acesso em: 15 jul. 2022.

30 GAD, A. F. **Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall**, 2021.Disponível em: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/. Acesso em: 14 jan. 2024.

31 NARKHEDE, S. **Understanding Confusion Matrix**, 2018. Disponível em: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. Acesso em: 14 jan. 2024.

32 FILHO, M. **Precisão, Recall e F1 Score Em Machine Learning**, 2023. Disponível em: https://mariofilho.com/precisao-recall-e-f1-score-em-machine-learning/#o-que-é-f1-score. Acesso em 14 jan. 2024.

33 GAD, A. F. **Evaluating Object Detection Models Using Mean Average Precision (mAP)**, 2021.Disponível em: https://blog.paperspace.com/mean-average-precision/. Acesso em: 14 jan. 2024.

34 HUI. J. **mAP (mean Average Precision) for Object Detection**, 2018. Disponível em: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. Acesso em: 17 jan. 2024.

35 ANDROID DEVELOPERS. **Conhecer o Android Studio**, 2023. Disponível em: https://developer.android.com/studio/intro?hl=pt-br. Acesso em: 14 jan. 2024.

36 PANCHAL, S. **Chaquopy: Using Python In Android Apps**, 2023. Disponível em: https://proandroiddev.com/chaquopy-using-python-in-android-apps-dd5177c9ab6b. Acesso em 14 jan. 2024.

37 BOESCH, G. **TensorFlow Lite – Real-Time Computer Vision on Edge Devices (2024)**, 2024. Disponível em: https://viso.ai/edge-ai/tensorflow-lite/. Acesso em 28 fev. 2024.

38 SIMPLILEARN. **What is MATLAB?**, 2023. Disponível em: https://www.simplilearn.com/tutorials/matlab-tutorial/what-is-matlab-introduction-for-beginners. Acesso em 14 jan. 2024.

39 GOOGLE. **Perguntas frequentes**, 201-?. Disponível em: https://research.google.com/colaboratory/intl/pt-BR/faq.html. Acesso em 15 jan. 2024.

40 ULTRALYTICS. **YOLOv5**, 2020. Disponível em: https://github.com/ultralytics/yolov5. Acesso em 15 jan. 2024.

41 GITHUB. **Aplicação de Reconhecimento de Placa Veicular**, 2024. Disponível em: https://github.com/sElton42/MobileAppPlateRecognition. Acesso em: 17 jan. 2024.

42 PAVITHRAN. P. **How to label custom images for YOLO – YOLO 3**, 2020. Disponível em: https://cloudxlab.com/blog/label-custom-images-for-yolo/. Acesso em 17 jan. 2024.

43 ARIE, L. G. **The practical guide for Object Detection with YOLOv5 algorithm**, 2022. Disponível em: https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843. Acesso em: 17 jan. 2024.

44 HANA. L. **An intro to Convolutional Neural Networks (CNN)**, 2019. Disponível em: https://lamiae-hana.medium.com/an-intro-to-convolutional-neural-networks-cnn-9f1c2d888fa1#:~:text=Normalization%20will%20help%20our%20algorithm,the%20class%20of%20an%20image. Acesso em 19 jan. 2024.

45 RIBEIRO, V. Z. **5 Passos para criar seu 1º Projeto de Deep Learning com Python e Keras**, 2021. Disponível em: https://www.insightlab.ufc.br/5-passos-para-criar-seu-1o-projeto-de-deep-learning-com-python-e-keras/. Acesso em: 28 fev. 2024.

46 WD DESENVOLVIMENTO. **API Placas**, 2022?. Disponível em: https://apiplacas.com.br/doc.php. Acesso em: 22 jan. 2024.

47 SERPRO. **Consulta Online Senatran**, 2022?. Disponível em: https://www.loja.serpro.gov.br/consultasenatran. Acesso em: 22 jan. 2024.

48 MONTEBUGNOLI, T. C. **Android - conhecendo a biblioteca Retrofit 2**, 2018. Disponível em: http://theclub.com.br/Restrito/Revistas/201811/ANDR1811.ASPX. Acesso em: 22 jan. 2024.

49 AKBARNEZHAD, E. **YOLOv8 Projects #1 "Metrics, Loss Functions, Data Formats, and Beyond"**, 2023. Disponível em: https://www.linkedin.com/pulse/yolov8-projects-1-metrics-loss-functions-data-formats-akbarnezhad/. Acesso em: 22 jan. 2024.

50 TUDOCELULAR. **Samsung Galaxy A32**, 2021. Disponível em: https://www.tudocelular.com/Samsung/fichas-tecnicas/n6707/Samsung-Galaxy-A32.html. Acesso em 26 fev. 2024.