



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
GRADUAÇÃO EM ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

Hiago Henrique dos Santos

**Desenvolvimento de um Sistema Embarcado de
Reconhecimento Automático de Placas Automotivas para
Gerenciamento de Tráfego.**

Patos de Minas - MG
2024

Hiago Henrique dos Santos

**Desenvolvimento de um Sistema Embarcado de
Reconhecimento Automático de Placas Automotivas para
Gerenciamento de Tráfego.**

Trabalho de Conclusão de Curso apresentado
ao Curso de Graduação em Engenharia Eletrônica
e de Telecomunicações da Faculdade de En-
genharia Elétrica da Universidade Federal de
Uberlândia como parte dos requisitos para a
obtenção do título de Bacharel em Engenharia
Eletrônica e de Telecomunicações.

Orientador: Prof. Dr. Rafael Augusto da
Silva.

Agradecimentos

Ao Prof. Dr. Rafael Augusto da Silva, minha sincera gratidão pela orientação dedicada e pelos preciosos ensinamentos transmitidos ao longo da elaboração desta monografia. Sua expertise e disponibilidade foram essenciais para o sucesso deste trabalho.

Aos meus queridos familiares, em especial aos meus pais e irmãos, expresso minha mais profunda gratidão pela compreensão, paciência e apoio incondicional ao longo de toda minha jornada acadêmica. Suas palavras de encorajamento e o suporte emocional foram fundamentais para que eu pudesse dedicar-me aos estudos com afinco e determinação.

Aos amigos que fiz durante a graduação, agradeço pela amizade sincera e pelo apoio mútuo, que foram imprescindíveis para superar os desafios e celebrar as conquistas ao longo deste percurso.

À Universidade Federal de Uberlândia (UFU), sou imensamente grato pela oportunidade de adquirir conhecimento e desenvolver habilidades que foram essenciais para a realização deste trabalho. Além disso, agradeço à universidade pela disponibilização de recursos e materiais que contribuíram significativamente para a elaboração deste projeto, permitindo que eu explorasse novas perspectivas e alcançasse resultados satisfatórios.

Ao Rayson Laroca por disponibilizar generosamente o acesso ao conjunto de dados UFPR-ALPR, que enriqueceu significativamente a pesquisa realizada para esta monografia. Sua disposição em compartilhar esse recurso valioso foi fundamental para avançar o escopo do trabalho, sua contribuição é profundamente apreciada.

Por fim, expresso minha gratidão a todas as pessoas que, direta ou indiretamente, contribuíram para o desenvolvimento desta monografia. Seu apoio e incentivo foram fundamentais para esta conquista, e por isso estou profundamente agradecido.

“Uma mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.”

(Albert Einstein)

Resumo

Os sistemas de Reconhecimento Automático de Placas Veiculares, ou *Automated License Plate Recognition* (ALPR), identificam e reconhecem o texto presente em placas veiculares a partir de suas imagens. Suas possíveis aplicações são amplas, incluindo fiscalização de trânsito, controle de acesso a áreas privadas ou monitoramento. Normalmente esse tipo de sistema já existe comercialmente, porém seus custos são, na maioria das vezes, muito altos, pois além de serem desenvolvidos com objetivo de render algum lucro, também exigem um hardware de alto desempenho para lidar com a grande quantidade de dados que o processamento digital de imagens provê. Este trabalho explora a viabilidade de utilização de hardware embarcado de baixo custo para cumprir o propósito de um sistema ALPR, utilizando um modelo de rede neural da TensorFlow Lite treinado especificamente para identificação de placas automotivas em imagens e técnicas de OCR, tanto embarcadas com a utilização da biblioteca Tesseract OCR quanto com a utilização de serviços em nuvem, utilizando o Vision AI. Também foram desenvolvidos algoritmos auxiliares para melhorar a acurácia do sistema como um todo, corrigindo possíveis erros advindos da etapa de OCR. Também foi construída uma interface utilizando a Custom TKinter para melhorar a visualização das etapas e facilitar as configurações do sistema. Os resultados obtidos para um conjunto de 30 imagens selecionadas de um banco de imagens são 93,25% de acurácia para o sistema operando em modo embarcado e 99,59% para o sistema operando com o serviço de OCR em nuvem.

Palavras-chave: Reconhecimento Automático de Placas Veiculares. ALPR. Reconhecimento de Caracteres. OCR. Redes Neurais.

Abstract

Automated License Plate Recognition (ALPR) systems consist of identifying and recognizing text on license plates, typically from images. The potential applications for such systems include traffic enforcement, access control to private areas, and monitoring. While this type of system already exists commercially, its costs are often high due to the need for high-performance hardware to handle digital image processing, in addition to being developed with the aim of making a profit. This work explores the feasibility of using low-cost embedded hardware to fulfill the purpose of an ALPR system, employing a TensorFlow Lite neural network model specifically trained for automotive plate identification in images and OCR techniques, both embedded using the Tesseract OCR library and cloud services using Vision AI. Auxiliary algorithms were also developed to improve the overall system accuracy by correcting potential errors arising from the OCR stage. Additionally, an interface was built using Custom TKinter to enhance the visualization of the steps and facilitate system configurations. The results obtained for a set of 30 selected images from an image database are 93.25% accuracy for the system operating in embedded mode and 99.59% for the system operating with cloud-based OCR service.

Keywords: Automatic License Plate Recognition. ALPR. Character Recognition. OCR. Neural Networks.

Lista de ilustrações

Figura 1.1 – Tamanho de mercado e taxa de crescimento anual (CAGR) de sistemas ALPR	15
Figura 2.1 – Modelos do antigo padrão nacional de placas veiculares.	20
Figura 2.2 – Modelos do padrão Mercosul de placas veiculares.	20
Figura 2.3 – Variações do padrão de placas Mercosul.	20
Figura 2.4 – Exemplos de placas com condições impróprias.	22
Figura 2.5 – Estrutura geral de um sistema ALPR.	23
Figura 2.6 – Exemplo de mudança de brilho em imagens.	24
Figura 2.7 – Exemplo do método de Média com validade de dados limitada em uma imagem.	24
Figura 2.8 – Exemplo de detecção de bordas em uma imagem.	26
Figura 2.9 – Exemplo de segmentação por limiar em uma imagem.	26
Figura 2.10–Exemplo de segmentação por região em uma imagem.	27
Figura 2.11–Classes de Redes Neurais Artificiais	29
Figura 2.12–Processo de convolução em uma matriz.	30
Figura 2.13–Exemplo estrutural de uma CNN.	31
Figura 2.14–Estágios da camada convolucional.	32
Figura 2.15–Exemplo de agrupamento por <i>Max pooling</i>	33
Figura 2.16–Estrutura de uma CNN destacando a camada totalmente conectada.	33
Figura 2.17–Componentes do sistema OCR	35
Figura 2.18–Conversão de imagem colorida para binária.	36
Figura 2.19–Processo de Segmentação de Caracteres.	36
Figura 3.1 – Imagens do banco original.	45
Figura 3.2 – Imagens adicionadas (Modelos Mercosul).	45
Figura 3.3 – Diagrama estrutural simplificado do sistema.	48
Figura 3.4 – Exemplo de uma GUI feita com a Custom TKinter.	49
Figura 3.5 – Imagem Limiarizada com valor fixo.	50

Figura 3.6 – Imagem limiarizada pelo método Gaussiano.	51
Figura 3.7 – Imagem limiarizada pelo método de Média.	51
Figura 3.8 – Processo de detecção de angulação pelo método do retângulo de área minima.	52
Figura 3.9 – Processo de detecção de angulação pelo método de Hough.	53
Figura 3.10–Método responsável por corrigir caracteres nos códigos.	55
Figura 3.11–Exemplo de confusão causada pelo OCR entre o número 2 e letra Z. . .	56
Figura 3.12–Exemplo de confusão causada pelo OCR entre o número 5 e letra S. . .	56
Figura 3.13–Exemplo de confusão causada pelo OCR entre o número 8 e letra B. . .	56
Figura 3.14–Exemplos de códigos corrigidos pelo algoritmo.	56
Figura 3.15–Método responsável por calcular a similaridade entre códigos.	57
Figura 3.16–Exemplo feito usando o algorítimo de calculo de similaridade (O e Q). .	59
Figura 3.17–Exemplo feito usando o algorítimo de calculo de similaridade (B e 8). .	59
Figura 3.18–Exemplo feito usando o algorítimo de calculo de similaridade (S e 5). .	60
Figura 4.1 – Gráfico de desempenho do treinamento do modelo de detecção.	63
Figura 4.2 – Gráfico de taxa de aprendizado do modelo de detecção.	64
Figura 4.3 – Camadas finais do modelo de rede neural.	65
Figura 4.4 – Interface principal do sistema desenvolvido.	66
Figura 4.5 – Painel lateral da interface.	67
Figura 4.6 – Painel inferior da interface.	68
Figura 4.7 – Interface de adição de códigos das placas autorizadas.	68
Figura 4.8 – Interface principal do sistema em funcionamento.	69
Figura 4.9 – Interface principal do sistema em funcionamento.	70

Lista de tabelas

Tabela 3.1 – Arquiteturas disponíveis para treinamento.	47
Tabela 3.2 – Correspondência entre números e letras	55
Tabela 4.1 – Métricas de Treinamento do Modelo	63
Tabela 4.2 – Tempo médio das etapas do sistema (Snapshot).	71
Tabela 4.3 – Tempo médio das etapas do sistema (Contínuo, máx. 5 segundos).	71
Tabela 4.4 – Acurácia média para uma captura única.	71
Tabela 4.5 – Acurácia média para uma captura contínua (máx. 5 segundos).	72
Tabela 4.6 – Acurácia média para uma captura contínua (máx. 5 segundos).	72

Lista de quadros

3.1	Matriz de Similaridade.	61
A.1	Dados de desempenho do sistema configurado para OCR em Nuvem.	82
A.2	Score de detecção e Acurácia do sistema configurado com OCR em nuvem.	83
A.3	Dados de desempenho do sistema configurado para OCR embarcado.	84
A.4	Score de detecção e Acurácia do sistema configurado com OCR embarcado.	85
B.1	Dados de desempenho do sistema configurado para OCR embarcado.	87
B.2	Score de detecção e Acurácia do sistema configurado com OCR embarcado.	88
B.3	Dados de desempenho do sistema configurado para OCR em nuvem.	89
B.4	Score de detecção e Acurácia do sistema configurado com OCR em nuvem.	90
C.1	Dados de desempenho do sistema configurado para OCR embarcado.	92
C.2	Score de detecção e Acurácia do sistema configurado com OCR embarcado.	93
C.3	Dados de desempenho do sistema configurado para OCR em nuvem.	94
C.4	Score de detecção e Acurácia do sistema configurado com OCR em nuvem.	95

Lista de siglas

API *Application Programming Interface*

AP *Average Precision*

AR *Average Recall*

ABESE Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança

ALPR *Automated License Plate Recognition*

ANPR *Automated Number Plate Recognition*

CAGR *Compound Annual Growth Rate*

CFTV *Closed-Circuit Television*

CNN *Convolutional Neural Network*

CONTRAN Conselho Nacional de Trânsito

DETRAN Departamento Estadual de Trânsito

GUI *Graphical User Interface*

GPU *Graphics Processing Unit*

HTTP *Hypertext Transfer Protocol*

JPL *Jet Propulsion Laboratory*

Mercosul Mercado Comum do Sul

MIT *Massachusetts Institute of Technology*

MRFR *Market Research Future*

OCR *Optical Character Recognition*

OpenCV *Open Source Computer Vision Library*

PDI Processamento Digital de Imagens

ReLU *Rectified Linear Unit*

RENAVAM Registro Nacional de Veículos Automotores

RNA Rede Neural Artificial

RNC Rede Neural Convolucional

SSD *Single Shot Detector*

TMR *Transparency Market Research*

TPU *Tensor Processing Unit*

UFPR Universidade Federal do Paraná

XML *Extensible Markup Language*

YOLO *You Only Look Once*

Sumário

1	INTRODUÇÃO	14
1.1	Tema do Projeto	16
1.2	Problematização	16
1.3	Hipótese	16
1.4	Objetivos	17
1.4.1	Objetivo Geral	17
1.4.2	Objetivos Específicos	17
1.5	Justificativas	17
1.6	Organização do Trabalho	18
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	Placas veiculares	19
2.2	Detecção de placas veiculares	21
2.3	Pré-processamento	23
2.4	Segmentação	25
2.5	Redes Neurais Artificiais - RNAs	27
2.5.1	Arquiteturas de Rede	28
2.5.2	Redes Neurais Convolucionais	29
2.6	Reconhecimento Óptico de Caracteres - OCR	34
2.6.1	Varredura Óptica	35
2.6.2	Segmentação de localização	36
2.6.3	Pré-processamento	37
2.6.4	Segmentação	37
2.6.5	Representação	37
2.6.6	Extração de recursos	38
2.6.7	Treinamento e Reconhecimento	38
2.6.8	Pós-processamento	40
2.7	Estado da arte	40

3	METODOLOGIA	44	
3.1	Construção do banco de imagens	44	
3.2	Treinamento do modelo de detecção de placas	46	
3.3	Desenvolvimento do sistema	47	
3.3.1	Estrutura do sistema	47	
3.3.2	Interface gráfica do sistema	49	
3.3.3	Algoritmos de detecção e segmentação das placas	49	
3.3.4	Algoritmos de processamento de imagem	50	
3.3.5	Algoritmo de OCR	53	
3.3.6	Algoritmo de correção de código	54	
3.3.7	Cálculo de similaridade entre códigos.	57	
3.3.8	Similaridade gráfica entre diferentes caracteres	58	
4	RESULTADOS	62	
4.1	Treinamento do modelo de detecção	62	
4.2	Interface Gráfica de Usuário (GUI)	66	
4.3	Desempenho do sistema	70	
4.4	Confiabilidade de detecção e acurácia do sistema	71	
5	CONSIDERAÇÕES FINAIS	73	
5.1	Trabalhos Futuros	74	
REFERÊNCIAS		75	
APÊNDICES		80	
APÊNDICE A	-	DADOS DE CAPTURA CONTÍNUA COM NO MÁXIMO 5 SEGUNDOS.	81
APÊNDICE B	-	DADOS DE CAPTURA ÚNICA SEM APLICAÇÃO DO ALGORITMO DE CORREÇÃO..	86
APÊNDICE C	-	DADOS DE CAPTURA ÚNICA COM APLICAÇÃO DO ALGORITMO DE CORREÇÃO..	91

CAPÍTULO 1

Introdução

Com o constante aumento no número de veículos nos pequenos e grandes centros urbanos, surge a necessidade de um sistema de reconhecimento de placas automotivas rápido e confiável. Com o uso de técnicas de Processamento Digital de Imagens (PDI), como as mostradas por Guingo, Rodrigues e Thomé (2002), é possível criar um sistema que além de detectar as placas automotivas, permita a identificação rápida e precisa de veículos suspeitos ou infratores. Ou ainda podem ser utilizados apenas para o controle de acesso a algum lugar restrito, como estacionamentos, condomínios ou áreas particulares.

De acordo com um relatório publicado pela *Transparency Market Research* (TMR), o mercado global de sistemas de reconhecimento automático de placas automotivas, ou *Automated License Plate Recognition* (ALPR), está previsto para atingir US\$ 1,4 bilhão até 2023, acompanhando um grande crescimento desde 2013, como mostrado na Figura 1.1. Segundo o relatório, os principais fatores que impulsionaram esse crescimento incluem o aumento na adoção de aplicativos de fiscalização e vigilância de tráfego e tendência de integração de sistemas ALPR com outros sistemas, como de câmeras de sistemas *Closed-Circuit Television* (CFTV). No entanto, fatores como a não uniformidade do design das placas estão restringindo o crescimento do mercado (InexTech, 2015; TMR, 2020).

Uma visão geral de mercado, dada pelo mesmo relatório, estima que a taxa de crescimento anual composta, ou *Compound Annual Growth Rate* (CAGR), para esse mercado seja de aproximadamente 9% de 2023 a 2030. Outro ponto é que o segmento governamental deve dominar o mercado durante esse mesmo período, enquanto o segmento comercial provavelmente será o segmento de crescimento mais rápido (TMR, 2020).

Figura 1.1 – Tamanho de mercado e taxa de crescimento anual (CAGR) de sistemas ALPR



Fonte: Adaptado de InexTech (2015, online).

Sistemas ALPR geralmente possuem duas partes: localização de placas de veículos e reconhecimento de caracteres, ou *Optical Character Recognition* (OCR). No estágio de localização, a placa automotiva é detectada e separada do restante da imagem/vídeo para reconhecimento de caracteres. Na próxima etapa, as imagens das placas são processadas e fornecidas como entrada para um mecanismo de reconhecimento de caracteres. Embora existam muitos sistemas ALPR em desenvolvimento, a maioria deles depende fortemente de recursos computacionais elevados, tornando-os complexos e caros (Batra et al., 2022).

Redes Neurais Convolucionais (RNCs), ou *Convolutional Neural Networks* (CNNs), são um método muito utilizado para a primeira etapa de um sistema ALPR, pois são muito eficientes na detecção de objetos em imagens. Muitos sistemas desses usam a bem-sucedida arquitetura *You Only Look Once* (YOLO), como apresentado por Xie et al. (2018). O problema dessas técnicas é que elas normalmente exigem toneladas de dados para serem treinadas do zero mesmo que a maioria dos métodos existentes explore o aprendizado por transferência, como mostrado por Yosinski et al. (2014), transferindo para redes menos complexas pesos de redes excessivamente treinadas (Silva; Jung, 2020).

Em seguida, com a detecção da placa feita, é necessário a identificação dos caracteres, com o uso de softwares, como o Tesseract OCR, um software de código aberto que atualmente é desenvolvido e mantido pela Google, ou com uso de outros pacotes, normalmente escritas em Python, como *Open Source Computer Vision Library* (OpenCV) e EasyOCR, sendo a OpenCV uma biblioteca de código aberto originalmente desenvolvida pela Intel, e a EasyOCR, criado por uma empresa chamada Jaided AI, escrito em Python com o uso da biblioteca PyTorch (Patel; Patel; Patel, 2012; Vedhavyassh et al., 2022).

1.1 Tema do Projeto

Este trabalho pretende desenvolver um sistema de Reconhecimento Automático de Placas Automotivas, ou ALPR, embarcado em um *hardware* de baixo custo, e portanto, poder computacional limitado, utilizando técnicas de Processamento Digital de Imagens (PDI), através de *Convolutional Neural Networks* (CNNs) e códigos de *Optical Character Recognition* (OCR), para controlar o acesso de veículos em ambientes restritos.

1.2 Problematização

Segundo a Associação Brasileira das Empresas de Sistemas Eletrônicos de Segurança (ABESE), o mercado nacional de segurança eletrônica teve um crescimento de 18% no ano de 2022, faturando aproximadamente 11 Bilhões de reais. Esses números refletem a renovação de produtos em projetos antigos e a atualização tecnológica causada pela implantação de novas tecnologias como sistemas eletrônicos de segurança (Abese, 2023).

O acesso a áreas restritas, como condomínios, imóveis particulares, estacionamentos privados, entre outros, normalmente carecem de algum tipo de controle para prover organização e segurança. Com isso em mente, e considerando o crescente numero de câmeras de videomonitoramento sendo instaladas tanto em propriedades públicas quanto em particulares, um sistema de reconhecimento automático de placas automotivas se torna uma ferramenta extremamente útil, identificando veículos por suas características, como o código de placa ou detalhes estéticos, permitindo ou não seu trânsito.

Normalmente sistemas como estes já existem comercialmente, porém seus custos são, na maioria das vezes, muito altos, pois além de possuírem o objetivo de render lucro, também exigem um *hardware* de alto desempenho para lidar com a grande quantidade de dados que o processamento digital de imagens provê. Portanto o desenvolvimento de sistemas como esse implantados em *hardwares* com menor poder computacional e de baixo custo é sempre bem-vindo.

1.3 Hipótese

Diante dos sistemas ALPR já existentes, sua baixa utilização em hardware embarcado está relacionada com as particularidades de implantação e características do dispositivo, como o poder computacional limitado.

Este trabalho explora a hipótese de que com estudos e pesquisas mais aprofundadas é possível viabilizar a utilização de *hardware* embarcado para atender os requisitos de um sistema ALPR.

1.4 Objetivos

1.4.1 Objetivo Geral

O presente trabalho possui como objetivo desenvolver um sistema de detecção automática de placas automotivas, embarcado em *hardware* de baixo custo, utilizando métodos de código aberto, para controle de acesso de automóveis em ambientes restritos.

1.4.2 Objetivos Específicos

- Realizar o levantamento bibliográfico relacionado a detecção automática de placas veiculares;
- Estruturar um banco de dados de imagens de placas veiculares para testes do sistema;
- Desenvolver o algoritmo de reconhecimento e extração de caracteres de placas automotivas em uma imagem/vídeo;
- Desenvolver o algoritmo de gerenciamento de acesso;
- Embarcar o sistema em um hardware de baixo custo.

1.5 Justificativas

A crescente necessidade por sistemas mais eficientes economicamente abre uma janela enorme para pesquisa e desenvolvimento. Sistemas ALPR com alta portabilidade e menor custo já se provam uma boa justificativa por proporcionar sua aplicação em diversas áreas, como por exemplo:

- Monitoramento e controle de tráfego;
- Identificação de veículos roubados;
- Cobrança automática de pedágio e estacionamento;
- Segurança e aplicação da lei e de multas;
- Controle de fronteiras e aeroportos.

Essas vantagens além de notáveis principalmente nas áreas de segurança e monitoramento, que são de extrema importância para qualquer país, também geram economia financeira, tornando-os muito mais escaláveis.

Além disso, sistemas como esse sempre necessitam de novos estudos e pesquisas pois novos métodos e tecnologias surgem, deixando os antigos métodos defasados. Esses pontos

em conjunto com a falta de materiais em português realçam ainda mais a importância do desenvolvimento de mais trabalhos na área.

Outro ponto importante é revelado considerando o crescimento do mercado de sistemas *Automated Number Plate Recognition* (ANPR), e consequentemente de sistemas ALPR. Segundo pesquisas do *Market Research Future* (MRFR) o mercado de reconhecimento de placas pode atingir o valor de US\$ 5,6 bilhões com um crescimento esperado de 9.5% até o ano de 2030, indicando uma carência do mercado por sistemas como estes (MRFR, 2020).

1.6 Organização do Trabalho

O presente trabalho está dividido em 5 capítulos. No Capítulo 1 é apresentada uma introdução a PDI, apontando fatos sobre sua história e sua importância, em sequência são introduzidos conceitos de reconhecimento de placas veiculares (ALPR), redes neurais e uma breve explicação do processo. No mesmo capítulo são apresentados o tema do projeto, a problematização, hipóteses, objetivos e justificativas.

O Capítulo 2 trata da fundamentação teórica do trabalho, descrevendo conceitos básicos necessários para a compreensão do trabalho como um todo, destacando Redes Neurais Artificiais (RNAs), Redes Neurais Convolucionais (RNCs) e *Optical Character Recognition* (OCR).

No Capítulo 3 são tratados detalhes de metodologia e materiais, descrevendo as técnicas de reconhecimento que serão utilizadas e materiais necessários para o funcionamento básico do projeto.

No Capítulo 4 são descritos os resultados do projeto, exibindo os resultados das etapas descritas na metodologia e os resultados de desempenho geral do sistema.

No Capítulo 5 é apresentada uma breve conclusão sobre o trabalho e seus resultados, juntamente com as dificuldades apresentadas no trabalho e possíveis melhorias para trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

2.1 Placas veiculares

Em 1886 os veículos como conhecemos foram inventados e poucos anos depois, a França, em 1893, foi a primeira nação a emitir uma placa veicular. Com uma popularidade explosiva, em 1901, Nova York foi o primeiro estado a exigir registro de automóveis motorizados, com isso os proprietários de automóveis foram obrigados a exibir as iniciais de seu nome na parte traseira do veículo em um local visível. O problema disso é que não havia conformidade, nenhuma restrição de estilo, cor ou mesmo materiais. Com o passar do tempo, o governo passou a emitir as placas e seus códigos, aumentando a padronização delas (Holm, 2022).

No Brasil o primeiro sistema de placas veiculares eram de responsabilidade das prefeituras e eram produzidas e emitidas nas cores preta ou vermelha e com caracteres na cor branca. Inicialmente se utilizava na primeira letra da placa a letra “P” quando o veículo era particular e a letra “A” para veículo de aluguel e a sequência numérica variava entre 1 a 99999. Esses sistemas passaram por muitas mudanças com o tempo mas em 1990 foi criado o Registro Nacional de Veículos Automotores (RENAVAM), permitindo com mais facilidade identificar cada automóvel registrado, criando também o sistema mais recente (AFAPEMG, 2019).

Atualmente no Brasil estão vigentes dois tipos de placas, um modelo antigo e um novo modelo baseado no padrão Mercado Comum do Sul (Mercosul), mostradas nas Figuras 2.1 e 2.2, respectivamente (CONTRAN, 2022).

Figura 2.1 – Modelos do antigo padrão nacional de placas veiculares.



Fonte: DobitaoByte (2017, online).

Figura 2.2 – Modelos do padrão Mercosul de placas veiculares.



Fonte: CONTRAN (2022, p. 4).

Além disso, segundo o Conselho Nacional de Trânsito (CONTRAN) o modelo de placas do Mercosul possuem algumas variações de cores segundo sua função, como mostrado na Figura 2.3 (CONTRAN, 2022).

Figura 2.3 – Variações do padrão de placas Mercosul.



Fonte: Diário do Transporte (2018, online).

Uma das características mais importantes das placas automotivas são seus códigos, e suas sequências não são totalmente aleatórias, essa característica pode ser utilizada para melhorar o desempenho do sistema de reconhecimento, utilizando técnicas que podem analisar o texto obtido e decidir se realmente é a melhor opção, aumentando ainda mais a acurácia do sistema. Esse método, também comentado na Seção 2.6.8, consiste em analisar o texto obtido e com base em dicionários ou dados probabilísticos de ocorrência de palavras ou caracteres, escolher qual é a melhor opção (Chaudhuri et al., 2017).

Quando houve a introdução do sistema nacional de registros baseado no RENAVAM, o Departamento Estadual de Trânsito (DETRAN) de cada unidade federativa recebeu certas sequências de três letras para registrar os veículos dentro da sua área de atuação, portanto cada estado possui sequências pré determinadas para seus veículos. Com isso um sistema que atua em um estado específico pode utilizar essas regras para ajudar a determinar os caracteres identificados que possuam pouca confiabilidade na detecção, ajudando, por exemplo, a diferenciar caracteres parecidos como 'Q' ou 'O' (Wikipédia, 2023; Chaudhuri et al., 2017)

Nessas tabelas sequenciais, cada unidade federativa possui uma ou mais sequências alfabéticas, de acordo com a demanda do estado por novos registros. Como exemplo, o estado do Paraná possui três diferentes sequências, sendo elas: AAA a BEZ, RHA a RHZ e SDP a SFO, portanto os veículos registrados no estado só podem possuir sequências dentro dessas séries (Wikipédia, 2023).

2.2 Detecção de placas veiculares

Detectar uma placa veicular de uma imagem pode aparentar ser uma tarefa fácil, mas existem muitos fatores que influenciam negativamente e sistemas como estes devem ser capazes de lidar com esses obstáculos. Como qualquer imagem, existem muitas variações que podem ocorrer durante sua captura, a localização da placa pode mudar para diferentes lugares da imagem, o tamanho da placa pode sofrer variação de acordo com a distância, pode ocorrer uma rotação nas imagens, tornando a imagem angulada, ou até mesmo uma oclusão parcial dela. Além desses citados, as condições de iluminação podem ser um grande problema, deixando sombras ou reflexos nas placas, dificultando ainda mais sua detecção, como mostrado pela Figura 2.4 (Jørgensen, 2017).

Figura 2.4 – Exemplos de placas com condições impróprias.

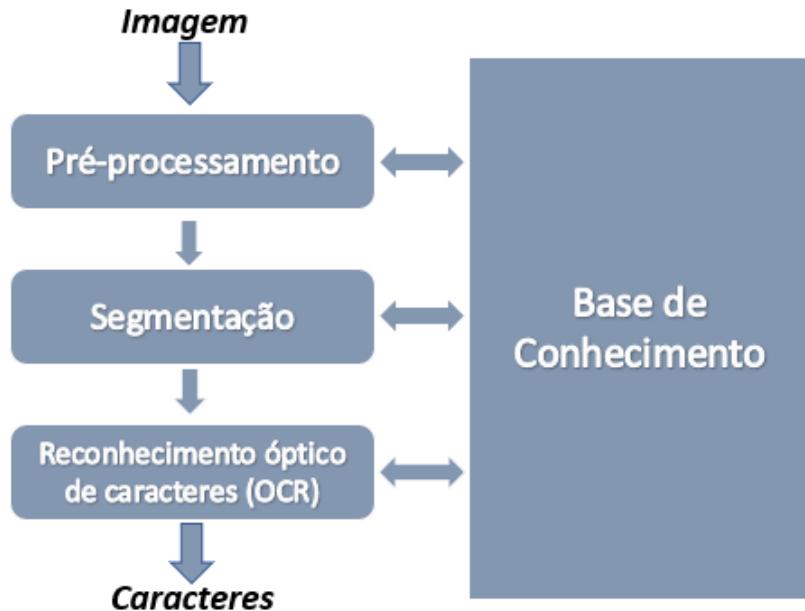


Fonte: Jørgensen (2017, p. 5).

Determinar a localização da placa é uma etapa de extrema importância para todo o sistema ALPR, além de reduzir a área em que os próximos processos serão aplicados, diminuindo assim os requisitos de processamento e por consequência o tempo, se algum erro ocorrer nessa etapa, todas as etapas posteriores podem ser parcialmente ou totalmente comprometidas, causando de um erro de reconhecimento de caracteres até nenhuma saída útil. Para isso existem diversos métodos, cada um com suas próprias características (FERNANDES, 2019).

Existe uma grande variação na organização de sistemas como esses, de forma geral, o algoritmo de um sistema ALPR pode ser subdividido em algumas etapas, como mostrado na Figura 2.5, cada uma delas possuem suas funcionalidades específicas que serão explicadas nas seções posteriores (Filho; Neto, 1999).

Figura 2.5 – Estrutura geral de um sistema ALPR.



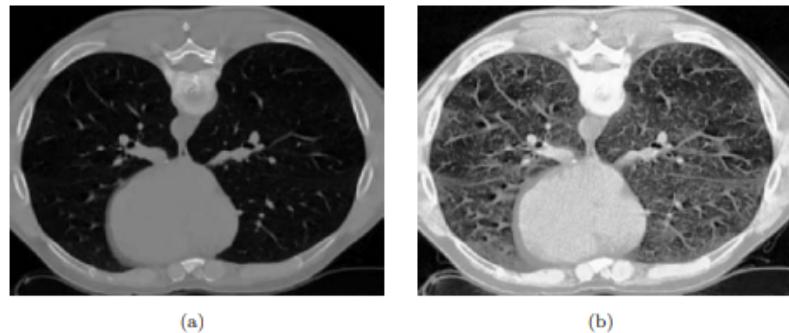
Fonte: Modificado de Filho e Neto (1999, p. 9).

2.3 Pré-processamento

A imagem de entrada resultante do processo de aquisição pode apresentar diversas imperfeições, como presença de pixels ruidosos, contraste e/ou brilho inadequados, caracteres interrompidos ou indevidamente conectados ou apresentar as condições impróprias como as mostradas na Figura 2.4. A etapa de pré-processamento possui a função de aprimorar a qualidade da imagem para as próximas etapas, uma vez que ajuda a suprimir informações irrelevantes para o processamento da imagem específica (Filho; Neto, 1999).

Existem diversos métodos de pré-processamento que podem ser subdivididos em diferentes grupos. Sonka, Hlavac e Boyle (2014) classificam esses métodos em quatro categorias: Transformações de brilho, Transformações geométricas, Pré-processamento Local e Restauração de imagem. De forma resumida, o método de transformação de brilho, exemplificado na Figura 2.6, modifica o valor de um pixel dependendo apenas das propriedades do próprio pixel, ou seja, a correção do brilho leva em consideração o seu brilho original e sua posição na imagem.

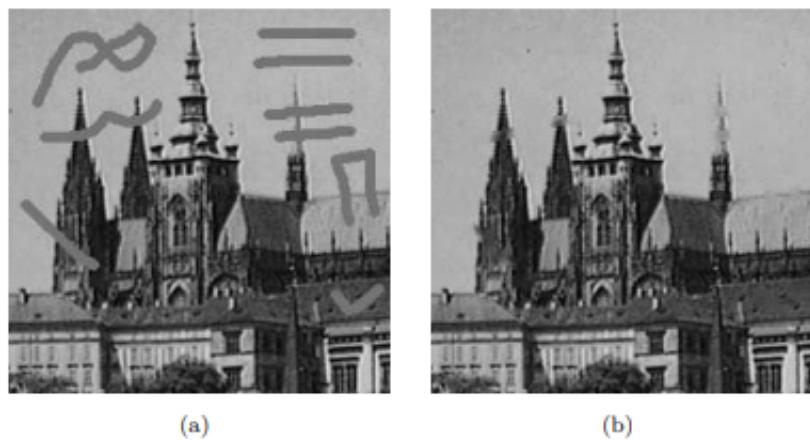
Figura 2.6 – Exemplo de mudança de brilho em imagens.



Fonte: Sonka, Hlavac e Boyle (2014, p. 120).

As transformações geométricas permitem eliminar distorções que ocorrem na aquisição das imagens, por exemplo, distorções causadas pela angulação da câmera em relação ao objeto. O método de pré-processamento local pode ser subdividido em dois grupos. O primeiro, chamado de suavização, visa suprimir ruídos ou outras pequenas flutuações na imagem, o equivalente, em termos de frequência, a suprimir altas freqüências. Um ponto negativo dessa suavização é o desfoque de bordas, que possuem informações importantes da imagem, um exemplo é mostrado na Figura 2.7, para um método suavização por média. O segundo grupo é chamado de operadores de gradiente, eles são baseados em derivadas locais, seus valores são maiores em locais da imagem que sofrem maiores variações, ou seja, no domínio da frequência é o equivalente a suprimir as baixas freqüências, realçando assim as bordas da imagem (Sonka; Hlavac; Boyle, 2014).

Figura 2.7 – Exemplo do método de Média com validade de dados limitada em uma imagem.



Fonte: Sonka, Hlavac e Boyle (2014, p. 129).

Por fim, os métodos de restauração de imagens visam suprimir a degradação usando o conhecimento sobre a natureza da imagem. A maioria desses métodos são baseados em convolução aplicada a imagem inteira. Esses métodos podem ser subdivididos em

determinísticos ou estocásticos. Os métodos determinísticos são aplicáveis a imagens com pouco ruido e uma função de degradação conhecida, aplicando então a função inversa para a correção da imagem. As técnicas estocásticas procuram encontrar a melhor restauração de acordo com algum critério estatístico particular, como por exemplo o método dos mínimos quadrados (Sonka; Hlavac; Boyle, 2014).

Vale ressaltar que todas as operações efetuadas nesta etapa são consideradas de baixo nível pois trabalham diretamente com os valores de intensidade dos píxeis, sem nenhum conhecimento sobre quais deles pertencem aos objetos, nesse caso, as placas (Filho; Neto, 1999).

2.4 Segmentação

A etapa de segmentação possui a tarefa básica de dividir a imagem em suas unidades significativas, ou seja, nos objetos de interesse que a compõem. Apesar de ser facilmente descrita, é uma das tarefas mais complexas e difíceis de ser implementada (Filho; Neto, 1999).

Dezenas de métodos e técnicas de segmentação de imagens são apresentados pelos autores Gonzalez e Woods (2010) e de Sonka, Hlavac e Boyle (2014), sendo as principais técnicas em destaque: Segmentação baseada em borda(*Edge-Based Segmentation*), Segmentação baseada em limiar(*Threshold-Based Segmentation*) e Segmentação baseada em região(*Region-Based Segmentation*) (Datagen, s.d.).

A Segmentação baseada em borda é uma das técnicas mais utilizadas no PDI, elas dependem de operadores de detecção de bordas que marcam as localizações de descontinuidades em níveis de cinza, cor, textura, ou outras nas imagens, um exemplo é mostrado na Figura 2.8. No entanto a imagem resultante da detecção de bordas não pode ser utilizada como resultado da segmentação. São necessárias etapas posteriores suplementares que devem combinar as bordas em cadeias que correspondem melhor às bordas da imagem. Outro ponto relevante é que nessa técnica, quanto mais informações prévias estiverem disponíveis para o processo, como formas e relações com outras estruturas na imagem, melhor serão os resultados (Sonka; Hlavac; Boyle, 2014).

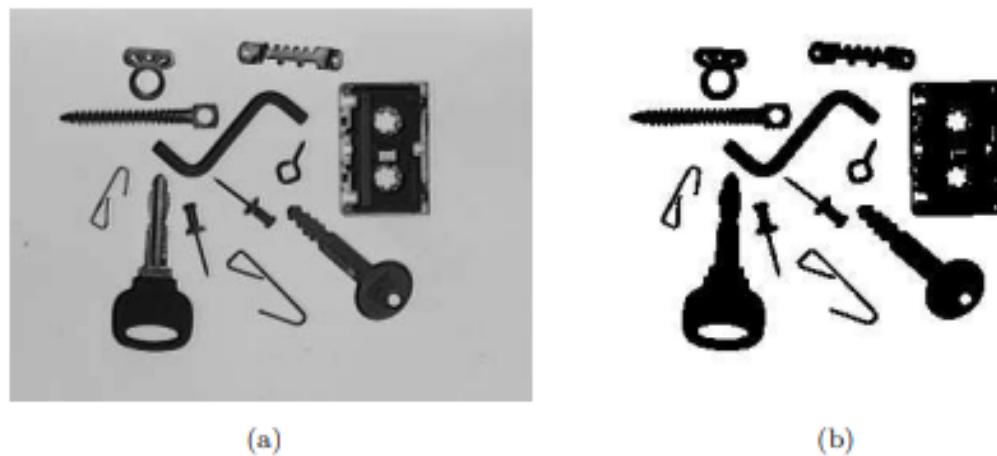
Figura 2.8 – Exemplo de detecção de bordas em uma imagem.



Fonte: Datagen (s.d., online).

O método mais simples e menos custoso computacionalmente é o de segmentação por limiar, nesse método uma imagem em tons de cinza é dividida pelos valores dos seus pixels, ou seja, ela é dividida baseando na intensidade dos pixels em relação a um valor chamado de limiar, um exemplo é mostrado na Figura 2.9. Esse valor de limiar pode ser uma constante ou ser dinâmico, variando dentro de uma mesma imagem (Datagen, s.d., online).

Figura 2.9 – Exemplo de segmentação por limiar em uma imagem.

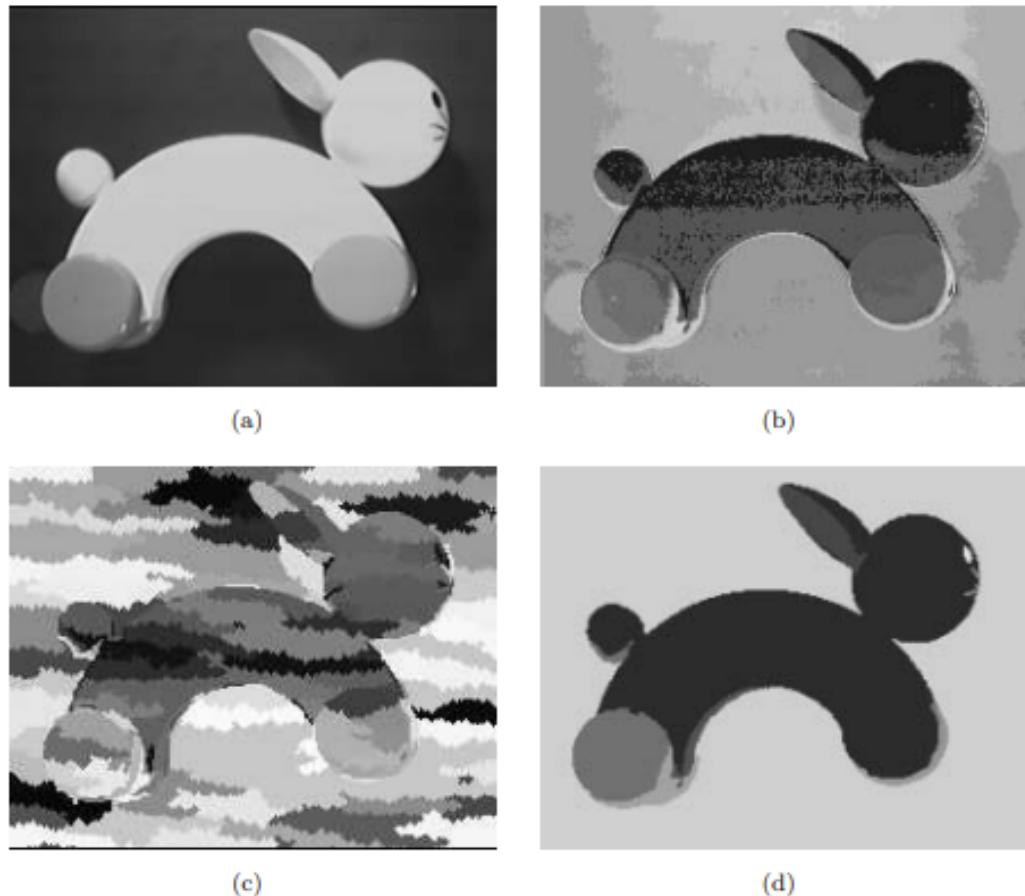


Fonte: Sonka, Hlavac e Boyle (2014, p. 180).

A ideia básica por trás da técnica de segmentação por região é a utilização de uma propriedade muito importante em imagens, a homogeneidade. Ela é utilizada como o principal critério de segmentação de regiões, onde a imagem é dividida em zonas de

máxima homogeneidade, em relação a níveis de cinza, cor, textura, formato, etc. Como mostrado pela Figura 2.10 (Sonka; Hlavac; Boyle, 2014).

Figura 2.10 – Exemplo de segmentação por região em uma imagem.



Fonte: Sonka, Hlavac e Boyle (2014, p. 224).

2.5 Redes Neurais Artificiais - RNAs

Projetar máquinas capazes de simular capacidades cognitivas humanas sem dúvida é uma área de pesquisa fascinante. Os neurônios são as entidades básicas do cérebro, suas conexões em redes permitem uma troca de informações que criam a inteligência biológica. Com esse conhecimento surge a ambição de tentar replicar suas estruturas em ambientes técnicos. Isso significa que a pesquisa tenta mapear a estrutura biológica em uma estrutura artificial, por exemplo com uma combinação de hardware e software, criando assim as Redes Neurais Artificiais (RNAs) (Rauber, 2005).

Segundo Rauber (2005) foram definidas uma quantidade extensa de modelos de RNAs e os métodos associados para adaptá-las às tarefas a serem resolvidas mas um fato é que os modelos artificiais tem pouco em comum com redes neurais reais, por outro lado existem paralelos entre esses dois mundos que prometem que as RNAs se aproximem para resolver

problemas cognitivos complexos.

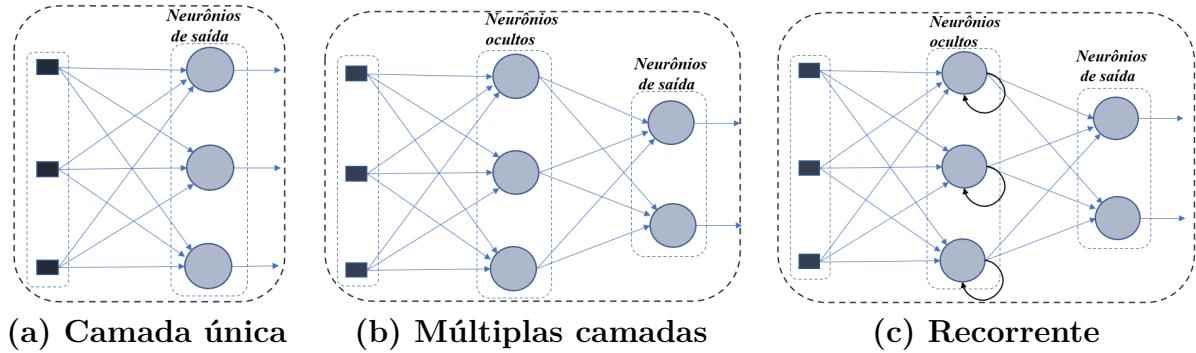
As RNAs possuem dois traços elementares: a arquitetura e o algoritmo de aprendizagem. Diferentemente de um computador com arquitetura Von Neumann que é programado, a rede é treinada por exemplos de treino pois o problema considerado está guardado dentro dos exemplos e devem estar obrigatoriamente disponíveis. O algoritmo de aprendizagem generaliza esses dados e memoriza esse conhecimento dentro dos parâmetros adaptáveis da rede, chamados de pesos sinápticos. De modo geral, uma RNA é uma máquina que é projetada para modelar a maneira com que o cérebro realiza uma tarefa particular ou função de interesse (Rauber, 2005; Haykin, 2001).

2.5.1 Arquiteturas de Rede

O modo com que os neurônios de uma RNA estão estruturados é intimamente ligado ao algoritmo de aprendizagem usado no treino da rede. Em geral, são identificados três classes de arquiteturas de rede. Segundo Haykin (2001), são elas:

- **Redes Alimentadas Adiante com Camada Única:** na sua forma mais simples, temos uma camada de nós fonte se projetando sobre uma camada de saída de neurônios, mas não vice-versa. Portanto ela é uma rede estritamente alimentada adiante, chamada também de rede Feedforward de camada única, ela é mostrada na Figura 2.11a ;
- **Redes Alimentadas Adiante de Múltiplas Camadas:** se distingue da primeira por possuir neurônios ocultos que possuem a função de intervir entre a entrada externa e a saída da rede. A adição de camadas torna a rede capaz de extrair estatísticas de ordem elevada, ou seja, em sentido livre, a rede adquire uma perspectiva global apesar de sua conectividade local. A Figura 2.11b mostra a estrutura dessa rede;
- **Redes Recorrentes:** também conhecidas por redes Feedback, ela se distingue das anteriores por possuir pelo menos um laço de realimentação, que se refere a uma situação onde a saída de um neurônio é redirecionada para sua própria entrada. Esses laços de realimentação tem um impacto profundo na capacidade de aprendizagem da rede, um exemplo estrutural dessa classe de rede é mostrada na Figura 2.11c.

Figura 2.11 – Classes de Redes Neurais Artificiais



Fonte: Autor.

A propriedade primordial das redes neurais citadas e outras é sua habilidade de aprender a partir do seu ambiente e melhorar seu desempenho através da aprendizagem. Para o contexto de redes neurais, o aprendizado pode ser definido como um processo pelo qual os parâmetros livres são adaptados através dos estímulos vindos do ambiente em que esta está inserida. O algoritmo de aprendizagem é a denominação para um conjunto de regras bem definidas pre estabelecidas para a solução de um problema de aprendizagem (Haykin, 2001).

Basicamente esses algoritmos de aprendizagem diferem entre si pela forma com que os ajustes dos pesos sinápticos são feitos. Segundo Haykin (2001) existem cinco regras básicas de aprendizagem: aprendizagem por correção de erros, aprendizagem baseada em memória, aprendizagem hebbiana, aprendizagem competitiva e aprendizagem de Boltzmann.

2.5.2 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (RNCs), do inglês *Convolutional Neural Networks (CNNs)*, são uma conhecida arquitetura de aprendizagem profunda inspirada no mecanismo de percepção visual natural dos seres vivos, elas são especializadas em processamento de dados com topologia semelhante a uma grade ou matriz, como por exemplo imagens. Seu nome vem de uma operação matemática chamada convolução, portanto CNNs utilizam a convolução no lugar da multiplicação geral da matriz em pelo menos uma de suas camadas (Goodfellow; Bengio; Courville, 2016; Gu et al., 2018).

A convolução é um operador matemático linear que resulta na representação matemática de como um sistema linear opera sobre um sinal, ou seja, a partir de duas funções, resulta em uma terceira que representa a soma do produto dessas funções ao longo da região de superposição delas em função do deslocamento entre elas. Sejam as funções f e g contínuas para a variável contínua t , a convolução pode ser definida pela Equação 1 (Ferreira, 2017).

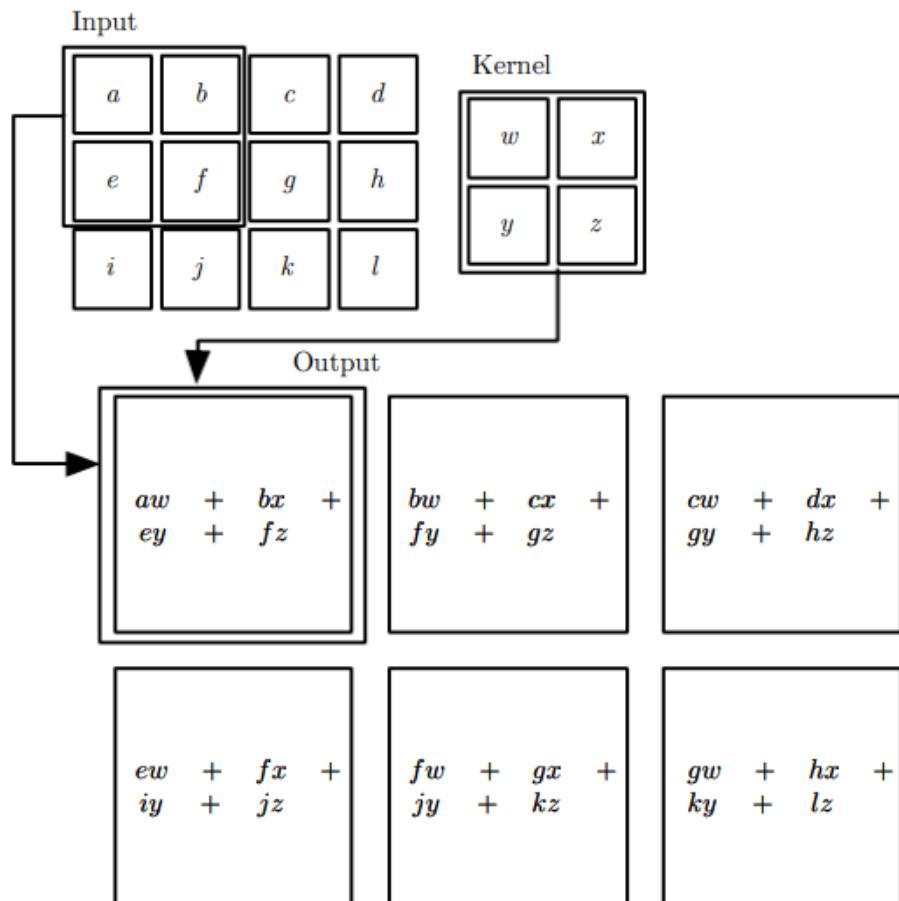
$$s(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (1)$$

Normalmente, quando se trabalha com dados em computadores, o tempo será discretizado, portanto a variável t só pode assumir valores inteiros. Então se as funções f e g forem definidas apenas para valores de t inteiros, temos a Equação 2 (Goodfellow; Bengio; Courville, 2016).

$$s(t) = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m] \quad (2)$$

A operação de convolução discreta, mostrada na Eq.2 pode ser interpretada como o somatório da multiplicação de cada elemento da matriz da imagem juntamente com seus vizinhos locais, por elementos de uma matriz que representa o filtro de convolução, também chamada de Kernel ou máscara, a Figura 2.12 exibe um exemplo desse processo (Ferreira, 2017).

Figura 2.12 – Processo de convolução em uma matriz.

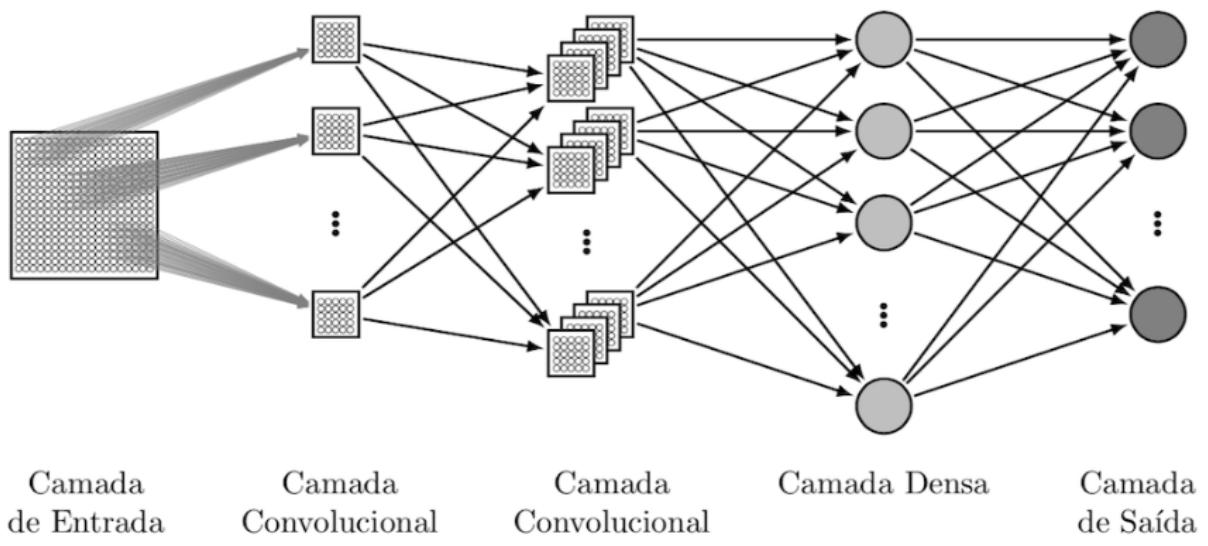


Fonte: Goodfellow, Bengio e Courville (2016, p. 330).

Nesse tipo de rede neural cada camada é composta por vários neurônios, cada um deles responsável por aplicar um filtro em uma parte específica da imagem. Os pesos atribuídos

às conexões entre neurônios da rede podem ser interpretados como o filtro convolucional ou kernel, com isso as CNNs passam a realizar a análise de campos receptivos locais (*local receptive fields*), um exemplo de CNN é mostrado na Figura 2.13 (Vargas; Paes; Vasconcelos, 2016).

Figura 2.13 – Exemplo estrutural de uma CNN.

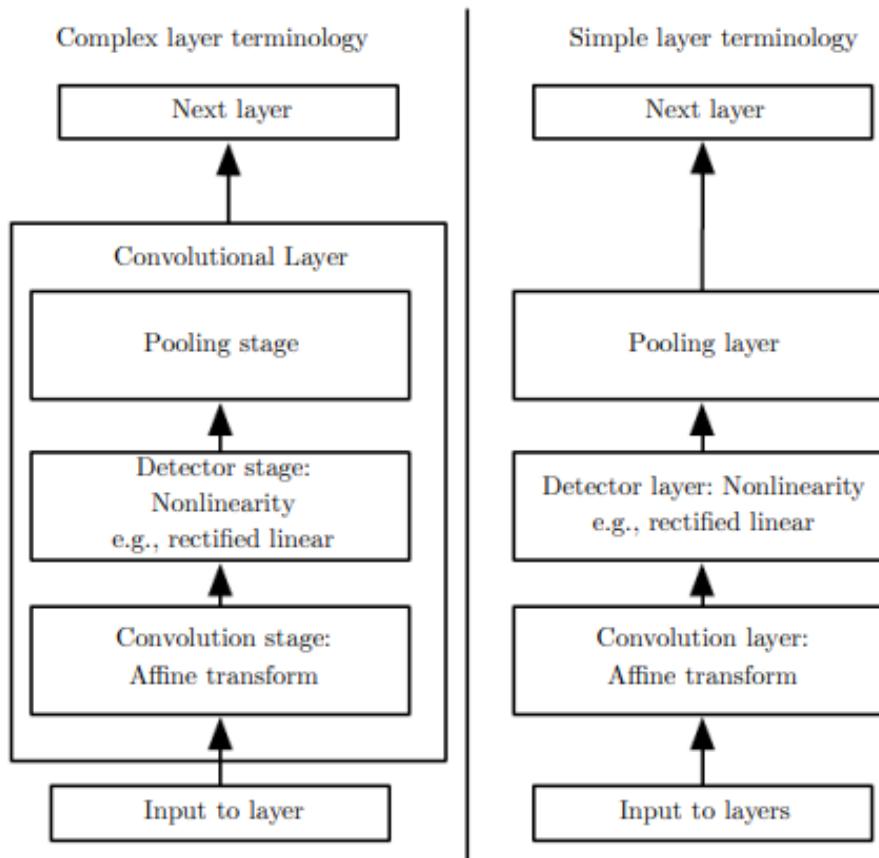


Fonte: Sakurai (2017, online).

Como dito por Vargas, Paes e Vasconcelos (2016), ao contrário de métodos tradicionais de visão computacional, onde é necessário definir um modelo de filtro a ser utilizado, nas CNNs somente é preciso definir a arquitetura dos filtros como quantidade, tamanho, *stride*(passo), dos filtros por camada. O processo de aprendizado da rede altera os pesos de acordo com seu treinamento para encontrar os melhores pesos sinápticos para o conjunto de dados de entrada analisados.

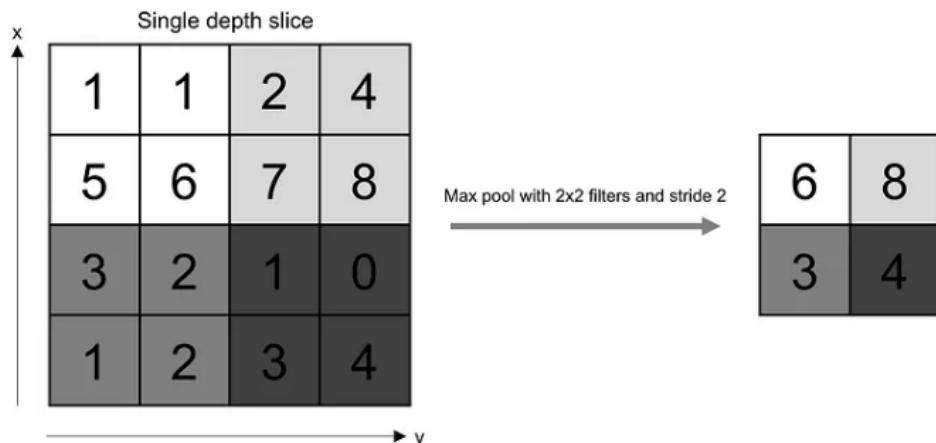
Uma camada convolucional típica consiste em três estágios (Figura 2.14). O primeiro estágio realiza várias convoluções em paralelo, ou seja, realiza o produto escalar entre duas matrizes, no qual uma delas é o Kernel, com parâmetros aprendíveis. No segundo estágio, também conhecido como estágio detector, uma função não linear é aplicada. Isso é feito para tornar as CNNs capazes de modelar também relações não-lineares, como imagens. Algumas funções não lineares mais comuns são: Sigmoid, Tanh e *Rectified Linear Unit* (ReLU) (Goodfellow; Bengio; Courville, 2016; Expert, 2020).

Figura 2.14 – Estágios da camada convolucional.



Fonte: Goodfellow, Bengio e Courville (2016, p. 336).

O terceiro estágio da camada convolucional é estagio de agrupamento (*Pooling*), também conhecido como *downsampling*, sua função é reduzir o conjunto de dados proveniente dos estágios anteriores. Semelhante à camada convolucional, a operação de agrupamento varre um filtro em toda a entrada, porém sem pesos, em vez disso a Kernel utilizada aplica uma função de agregação, que pode ser *Max pooling* (Figura 2.15), que seleciona o pixel com o valor máximo para enviar para a matriz de saída, *Average pooling*, que calcula o valor médio, ou outros métodos. (Mishra, 2020; Moura, 2021).

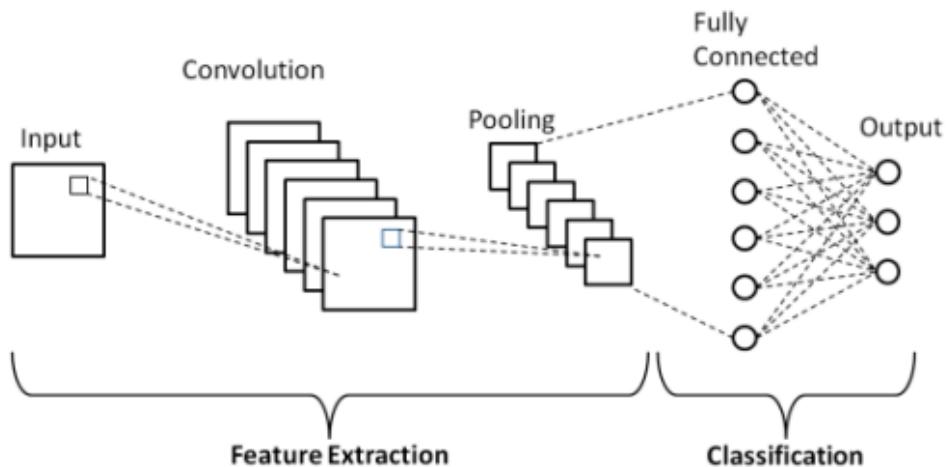
Figura 2.15 – Exemplo de agrupamento por *Max pooling*.

Fonte: Mishra (2020, online).

Em todos os casos, o estágio de agrupamento produz uma certa invariância de translação (*translation invariance*), isso significa que se a entrada sofrer um pequeno deslocamento a CNN ainda produzirá os mesmos resultados. Essa propriedade pode se tornar muito útil quando se preocupa mais se algum recurso está presente do que exatamente onde ele se encontra, ou seja, o objeto seria reconhecível independentemente da posição dele no quadro (Goodfellow; Bengio; Courville, 2016)(Mishra, 2020).

A ultima etapa de uma CNN, logo após as camadas convolucionais, é a camada totalmente conectada ou *Fully Connected Layer*(Figura 2.16), ela é responsável pela classificação com base nos recursos extraídos pelas camadas anteriores, essa camada, diferentemente das convolucionais que tendem a utilizar funções ReLU, geralmente utilizam uma função de ativação softmax (função exponencial normalizada) para classificar as entradas apropriadamente (Mishra, 2020)(IBM, s.d.)

Figura 2.16 – Estrutura de uma CNN destacando a camada totalmente conectada.



Fonte: Phung e Rhee (2019, p. 3).

2.6 Reconhecimento Óptico de Caracteres - OCR

Com o grande avanço tecnológico presenciado nas últimas décadas, o PDI vem sendo objeto de um crescente interesse por permitir e viabilizar um grande número de aplicações, subdivididas principalmente em duas categorias bem distintas: melhora das informações visuais para a interpretação humana e processamento de dados de imagens para transmissão, armazenamento e representação, considerando a percepção automática por máquinas (Gonzalez; Woods, 2010).

A história do PDI remonta ao final da década de 60, quando os primeiros sistemas de processamento digital de imagens foram criados com base em técnicas desenvolvidas pelo *Jet Propulsion Laboratory* (JPL), *Massachusetts Institute of Technology* (MIT) e outros. Naquela época, os sistemas eram grandes, caros e usavam computadores mainframe para processar as imagens capturadas por câmeras. Com o avanço da tecnologia, os sistemas de processamento digital de imagens evoluíram rapidamente, tornando-se mais acessíveis e eficientes. Com esse avanço e com o surgimento de novas linguagens de programação, houve uma completa transformação no processamento de imagens, possibilitando a criação de algoritmos mais complexos e sistemas mais precisos. Hoje em dia, o processamento digital de imagens está presente em várias áreas da vida, como na medicina, engenharia, entretenimento, segurança pública e muitas outras (Gonzalez; Woods, 2010; Weibel, 1999).

Apesar de que a visão humana seja incrivelmente poderosa e versátil, permitindo-nos perceber e compreender uma grande variedade de informações visuais em tempo real, quando se trata de tarefas específicas, a visão computacional, uma sub-área do PDI, possui diversas vantagens. Uma delas é a sua capacidade de processar grandes quantidades de dados de maneira rápida, precisa e com consistência e objetividade. Diferentemente do cérebro humano, que pode levar alguns segundos, ou pior, pode ser influenciado por fatores como fadiga, distração e emoção (Marengoni; Stringhini, 2009; Crosta, 1999).

Além disso a visão computacional pode ser facilmente integrada com outras tecnologias e sistemas, permitindo a automação de tarefas e a criação de soluções mais eficientes e inteligentes. Isso é especialmente útil em áreas onde a visão computacional é essencial para a navegação, reconhecimento de objetos, interação com o ambiente e gerenciamento (Gonzalez; Woods, 2010).

O reconhecimento óptico de caracteres permite o reconhecimento automático de caracteres através de um mecanismo óptico, como câmeras e scanners. Assim como nos seres humanos, nossos olhos são o mecanismo óptico que capturam imagens que servem de entrada para o cérebro. O OCR, embora ainda não possa competir com a capacidade humana de leitura, tenta imitá-la com o maior nível de confiabilidade. Esses sistemas podem reconhecer texto impresso ou manuscrito, porém seu desempenho está diretamente dependente da qualidade da imagem ou documento de entrada (Mithe; Indalkar; Divekar,

2013).

Para o reconhecimento automático de padrões, o principal princípio é primeiro ensinar à máquina quais classes de padrões podem ocorrer e como cada um deles se parece. No OCR esses padrões são normalmente letras, números ou símbolos específicos como vírgulas, pontos e outros. Esse ensinamento é feito mostrando à máquina exemplos de personagens de todas as diferentes classes. Com base nisso, ela constrói um protótipo ou uma descrição de cada uma delas. Por fim, durante o reconhecimento, os caracteres desconhecidos são comparados com as descrições e atribuídos à classe que possuir maior correspondência (Eikvil, 1993).

Tipicamente, sistemas OCR consistem em vários componentes que atuam separadamente, digitalizando uma imagem, localizando e segmentando seus caracteres até determinar a identidade de cada símbolo com base em comparações de características dos símbolos, a Figura 2.17 mostra a organização de um sistema (Eikvil, 1993).

Figura 2.17 – Componentes do sistema OCR



Fonte: Adaptado de Chaudhuri et al. (2017).

2.6.1 Varredura Óptica

O primeiro componente de um sistema OCR é a digitalização óptica, através de um dispositivo sensor que converte a intensidade da luz em dados digitais, que posteriormente são convertidos em níveis de cinza. Essa imagem multinível é convertida em uma imagem preto e branco de apenas dois níveis, comumente chamada de imagem binária. Esse processo, conhecido como limiarização, determina quais pixels são pretos e quais são brancos. Para isso, um limite pode ser pré-definido, fazendo com que pixels acima dele sejam pretos e abaixo sejam brancos, um exemplo dessa conversão é mostrado na Figura 2.18 (Chaudhuri et al., 2017).

Figura 2.18 – Conversão de imagem colorida para binária.



No entanto, imagens obtidas na prática necessitam de métodos mais sofisticados para determinar o limite e assim obter os melhores resultados. Nos melhores métodos o limite é variado de acordo com as propriedades locais da imagem, como o contraste e brilho, que por outro lado exigem mais capacidade computacional (Chaudhuri et al., 2017).

2.6.2 Segmentação de localização

O próximo componente do sistema OCR é a segmentação de localização, mostrada na Figura 2.19. É essa etapa que determina os constituintes da imagem, ou seja, a segmentação é o isolamento dos caracteres ou palavras. Normalmente essa etapa é realizada isolando cada um dos caracteres, no entanto alguns problemas podem surgir, como a extração de caracteres que estão em contato, a distinção entre ruído e texto e a má interpretação de gráficos ou geometrias com texto (Chaudhuri et al., 2017).

Figura 2.19 – Processo de Segmentação de Caracteres.



Fonte: Adaptado de Barreto (2018).

2.6.3 Pré-processamento

O terceiro componente OCR é o pré-processamento, nessa etapa, dependendo do tipo de aquisição dos dados, vários estágios de processamento preliminares são feitas para tornar os dados utilizáveis nos estágios seguintes. A imagem resultante da digitalização pode conter ruído, deixando os caracteres borrados ou quebrados, causando uma baixa taxa de reconhecimento. Nessa etapa são aplicados alguns métodos de PDI como preenchimento e erosão, deixando os caracteres mais suaves, e variantes da transformada de Hough, para detectar e corrigir distorções (Chaudhuri et al., 2017).

Os principais objetivos dessa etapa são a redução do ruído, normalização dos dados e compressão na quantidade de informação a ser processada. A redução de ruído é importante para reduzir segmentos de linhas desconectados, saliências e lacunas nas linhas, etc. A normalização visa retirar as variações da escrita e obter dados padronizados. Por fim na compressão para OCR são usadas técnicas no domínio espacial para preservar as informações de forma dos caracteres (Chaudhuri et al., 2017).

2.6.4 Segmentação

Aqui a imagem é segmentada em seus subcomponentes, a segmentação é importante porque a extensão que se pode alcançar na separação das várias linhas nos caracteres afeta diretamente a taxa de reconhecimento. As estratégias de segmentação de caracteres pode ser divididas em três categorias: segmentação explícita, segmentação implícita e estratégias mistas (Chaudhuri et al., 2017).

Na abordagem de segmentação com base explícita ou segmentação externa, os segmentos são identificados com base em propriedades semelhantes a caracteres. Este processo de cortar a imagem em componentes significativos recebe um nome especial, dissecação, onde ocorre uma análise do caráter sem usar uma classe específica. A estratégia de segmentação implícita, ou segmentação interna, é baseada no reconhecimento, procurando por componentes que correspondam às classes previamente definidas (Rehman; Mohammad; Sulong, 2009).

As técnicas mistas utilizam uma combinação das técnicas explícita e implícita, onde um algoritmo de dissecação é aplicado à imagem, mas a intenção é cortar a imagem em muitos lugares de forma que os limites de segmentação corretos sejam incluídos entre os cortes feitos. Com isso busca-se a segmentação ótima por meio da avaliação dos subconjuntos (Chaudhuri et al., 2017).

2.6.5 Representação

A representação da imagem desempenha um dos mais importantes papéis em qualquer sistema de reconhecimento. Em casos mais simples, as imagens em tons de cinza ou binárias são enviadas diretamente para um reconhecedor, no entanto, na maioria dos

sistemas, para evitar complexidade e aumentar a precisão, uma representação mais compacta é utilizada. Isso é feito extraindo um conjunto de características para cada classe, ajudando a distingui-las (Chaudhuri et al., 2017).

Nesse componente, geralmente, os métodos de representação de imagens são categorizados em três grandes grupos: transformação global e expansão em série, representação estatística e representação geométrica e topográfica (Chaudhuri et al., 2017).

Na transformação global e expansão em série um sinal continuo, é considerado que um sinal continuo ou discreto possua mais informação do que precisa ser representado para fins de classificação. Uma maneira de representar esses sinais é por combinação de uma série de funções mais simples e bem definidas. Os coeficientes da combinação fornecem uma codificação mais compacta, chamada de transformação ou expansão em série (Chaudhuri et al., 2017).

A representação estatística de uma imagem por distribuição estatística de pontos cuida das variações de estilo até certo ponto. Embora esse tipo de representação não permita a reconstrução da imagem original, ela é utilizada para reduzir a dimensão do conjunto de recursos proporcionando alta velocidade e baixa complexidade (Arica; Yarman-Vural, 2001).

Na representação geométrica e topológica as propriedades globais e locais dos caracteres podem ser representadas por características geométricas e topológica com alta tolerância a distorções e variações. Além disso nesse tipo de representação é possível codificar algum tipo de conhecimento sobre a estrutura do carácter ou fornecer algum conhecimento sobre a composição do objeto (Arica; Yarman-Vural, 2001).

2.6.6 Extração de recursos

O objetivo da extração de recursos é capturar as características mais essenciais dos símbolos. A maneira mais direta de descrever um carácter é por uma imagem raster, que são imagens que contêm a descrição de cada pixel. Pode se também extrair certos aspectos que caracterizam o símbolo. Além disso a extração de recursos possui outra tarefa importante, a classificação, que consiste em identificar cada carácter e atribuir a ele a classe de carácter correta (Arica; Yarman-Vural, 2001).

2.6.7 Treinamento e Reconhecimento

Os sistemas OCR usam frequentemente metodologias de reconhecimento de padrões, que atribuem uma amostra desconhecida a uma classe predefinida. Numerosas técnicas para OCR podem ser investigadas em quatro abordagens gerais, como mostrado em Jain, Duin e Mao (2000), são elas: correspondência de modelos, técnicas estatísticas, técnicas estruturais e Redes Neurais Artificiais (RNAs) (Arica; Yarman-Vural, 2001).

Em todas essas abordagens, as técnicas de OCR usam estratégias analíticas ou holísticas para o treinamento e reconhecimento. A estratégia holística emprega uma abordagem de cima para baixo (*Top-Down*) para reconhecer o carácter completo, eliminando assim problemas de segmentação, contudo isso gera uma restrição de vocabulário e uma menor precisão de reconhecimento. Já as estratégias analíticas utilizam uma abordagem de baixo para cima (*Bottom-Up*), iniciando no nível do traço ou carácter indo até a produção de texto significativo. Nessa abordagem algoritmos de segmentação são necessários, adicionando complexidade e introduzindo erros de segmentação no sistema (Chaudhuri et al., 2017).

A técnica de correspondência de modelo é baseada na correspondência de modelos armazenados com o carácter a ser reconhecido. De modo geral, a operação de correspondência diz o grau de semelhança entre dois vetores, como grupo de pixeis, formas ou curvaturas em um espaço de recursos. Essas técnicas podem ser classificadas em três classes: pareamento direto, moldes deformáveis e pareamento elástico e pareamento por relaxação (Chaudhuri et al., 2017).

A teoria estatística da decisão preocupa-se com funções estatísticas de decisão e um conjunto de critérios de otimização, que maximizam a probabilidade do padrão observado dado o modelo. Essas técnicas são baseadas principalmente em três suposições: o conjunto possui uma distribuição gaussiana, ou uniforme, no pior dos casos, existem estatísticas suficientes disponíveis para cada classe e dado um conjunto de imagens $\{I\}$ é possível extrair um conjunto de atributos $\{f_i\} \in F; i = \{1, \dots, n\}$ que representa cada classe de padrões (Arica; Yarman-Vural, 2001).

Técnicas estruturais utilizam uma descrição recursiva de um padrão complexo em termos de padrões mais simples baseados na forma do objeto. Os caracteres são representados como uma união de seus primitivos estruturais, normalmente essa técnica emprega métodos gramaticais e métodos gráficos (Chaudhuri et al., 2017).

As técnicas utilizando Redes Neurais Artificiais, tratadas separadamente na Seção 2.5, aproveitam-se da arquitetura massivamente paralela das RNAs, possibilitando a execução a uma taxa mais alta em comparação a técnicas clássicas. Ela se adapta às mudanças nos dados e aprende características do sinal de entrada. Existem várias abordagens para o treinamento de RNAs, como correção de erros, Boltzman, Hebbian e aprendizado competitivo, que cobrem entrada de valores binários e contínuos, bem como aprendizado supervisionado e não supervisionado. Além disso, as arquiteturas de RNAs podem ser divididas em *Feedforward*(diretas) ou *Feedback*(recorrentes), sendo as redes Perceptron multicamadas as redes *Feedforward* mais utilizadas e as redes Neurais de Kohonen as redes *Feedback* mais utilizadas (Arica; Yarman-Vural, 2001).

2.6.8 Pós-processamento

Comumente o componente de pós-processamento trata de atividades como agrupamento e detecção e correção de erros. Até esse ponto nenhuma informação semântica é considerada. A forma mais simples de incorporar informações de contexto é a utilização de um dicionário para corrigir pequenos erros. A ideia básica é de verificar a ortografia de saída e fornecer alternativas para saídas não reconhecidas no dicionário, alterando a palavra para outra com a maior relação de semelhança. A desvantagem dos métodos de dicionário é que as pesquisas e comparações consomem muito tempo e recurso. Portanto outra técnica pode ser utilizada, utilizando a análise de possibilidade de caracteres aparecerem em sequência, para isso é utilizado regras que definem a sintaxe da palavra (Arica; Yarman-Vural, 2001; Chaudhuri et al., 2017).

2.7 Estado da arte

Diversas abordagens já foram apresentadas para a identificação de placas veiculares feitas por software. Há trabalhos que empregam sistemas robustos, com computadores modernos e alguns poucos que se arriscam implementado em sistemas mais carentes de recursos. Em relação ao sistema de captação, também existem muitas variações, enquanto alguns utilizam câmeras simples, outros utilizam câmeras com alta resolução e alta taxa de atualização. Ainda existem aqueles que utilizam apenas imagens ou vídeos gravados para comprovar seus sistemas.

Além disso, a finalidade do sistema também pode variar bastante, indo desde apenas um reconhecimento até sua utilização para controle de pedágios, cancelas e outros. Para isso são utilizados uma grande variedade de métodos diferentes, cada um com suas particularidades. Abaixo se encontra um pequeno mapeamento de outros trabalhos relacionados ao tema, começando pelos mais recentes.

O artigo de Natalya e Sergei (2022) propõe um software para reconhecimento de placas de licença de veículos. O software é projetado para integração em um módulo autônomo instalado em um portão na entrada de uma área protegida. Uma característica do software é o uso de algoritmos de visão computacional, cuja implementação requer recursos computacionais mínimos, pois o fluxo de vídeo é processado por uma CPU de baixo desempenho. O software é executado na forma de uma biblioteca em linguagem C++ com o uso de funções padrão da biblioteca de visão computacional OpenCV. O software consiste nos seguintes passos: pré-processamento e binarização da imagem, localização da placa de licença, rotação e normalização da placa, segmentação dentro da placa de licença, validação do resultado da segmentação e reconhecimento de texto.

Para verificar e testar o software, uma câmera foi instalada no portão com o subsequente processamento dos dados de vídeo recebidos. Como resultado dos testes, obteve-se uma probabilidade de reconhecimento correto de 0.96, probabilidade de erro de reconhe-

cimento de 0.004, probabilidade de falha de 0.035 e probabilidade de reconhecimento falso de 0.015. O tempo de processamento de cada quadro, com resolução de 3 MPix em uma CPU Orange Pi Pc 2 com processador Allwinner H5 Quad-Core ARM Cortex-A53 de 64 bits, é de 1.2 a 1.5 segundos (Natalya; Sergei, 2022).

O trabalho de Valdeos et al. (2022) teve como objetivo projetar um sistema de reconhecimento de placas de licença peruano para reduzir o tempo de registro de veículos. Utilizando técnicas de processamento de imagem e a biblioteca OpenCV, juntamente com uma rede neural YoloV4, foi possível localizar com precisão a área onde a placa de licença está localizada. Isso facilitou a aplicação de um Reconhecedor Óptico de Caracteres (OCR) para extrair as informações de registro da placa.

Os resultados mostraram uma melhoria significativa em relação ao software tradicional de placas de licença. Utilizando um novo banco de dados peruano e avaliando com 200 a 1000 imagens, o sistema alcançou uma precisão de 100% e uma taxa de falha de 0% na detecção das placas de licença. Além disso, a sensibilidade e especificidade do sistema foram de 100% (Valdeos et al., 2022).

No entanto, no caso da rede treinada com 800 imagens, foi obtida uma precisão de 92% e uma exatidão de 95%, pois em 5 imagens ocorreram falsos positivos (considera-se falso positivo a detecção equivocada de uma área que não é uma placa, a detecção parcial da placa e a detecção dupla da mesma placa). À medida que a quantidade de imagens aumenta, são observadas melhorias marginais, atingindo o valor máximo de 100% (Valdeos et al., 2022).

É comentado no artigo que tempos de detecção são consideravelmente menores usando o Google Colab, com um tempo médio de 200 ms, enquanto no CMD do Windows chega a 1500 ms. Esse tempo é inversamente proporcional à capacidade computacional do computador. No Google Colab, o tempo permanece estável devido ao uso da GPU na nuvem. Por fim, é recomendado usar CNN em vez do "Tesseract" como mecanismo de reconhecimento (Valdeos et al., 2022).

Esses resultados indicam que o sistema de reconhecimento de placas de licença peruano desenvolvido neste trabalho é altamente eficaz e pode ser aplicado para obter informações de registro de veículos de forma rápida e precisa. Além disso, o banco de dados criado pode ser utilizado em futuros trabalhos relacionados aos veículos peruanos (Valdeos et al., 2022).

O trabalho de Moura (2021) compartilha do mesmo objetivo do trabalho de Filho (2019), controlar o acesso a um condomínio. Porém são utilizados outros métodos, para a identificação da placa foi utilizado um modelo modificado de rede pré-treinado do *TensorFlow*, chamado de *MobileNet-SSD*. Para detecção dos caracteres utilizou-se métodos de OCR, juntamente com CNNs. A implementação do algoritmo foi realizada em Python. A rede neural convolucional profunda apresentou bons resultados de precisão, revocação e F1-score, métricas de desempenho de redes neurais.

Além disso o trabalho também utiliza o reconhecimento facial como parâmetro para a permissão de acesso, para isso foi empregado o histograma de gradientes orientados alimentando uma máquina de vetor de suporte utilizando as bibliotecas *dlib* e *face_recognition*. Para a identificação e reconhecimento de placas de carros, o algoritmo implementado detecta e reconhece bem a maioria das placas dos carros. No entanto, erros de reconhecimento foram observados em algumas imagens sobre teste. Os resultados obtidos da avaliação do modelo foram a precisão de 61,13% e revocação de 62,65% (Moura, 2021).

Com relação a identificação e reconhecimento de rostos de pessoas, foi realizado o balanceamento da base de dados de imagens da *Labeled Faces in the Wild*, que permitiu obter resultados coerentes de predição do algoritmo. Também foi realizado um estudo do limiar que apresenta os melhores resultados das métricas de desempenho. Para o limiar de 0.18, obteve-se acurácia de 97,7%, precisão de 98,8%, revocação de 99,1% e F1-score de 99,1 % (Moura, 2021).

O artigo de Laroca e Menotti (2020) apresenta um sistema eficiente e robusto de Reconhecimento Automático de Placas de Veículos (ALPR) baseado no detector de objetos YOLO. O sistema proposto possui uma abordagem unificada para detecção de placas e classificação de layout, eliminando uma limitação comum em sistemas ALPR existentes, que é depender do layout das placas. Além disso, é introduzido um conjunto de dados publicamente disponível para ALPR, chamado UFPR-ALPR, que se tornou muito popular, sendo baixado mais de 650 vezes por pesquisadores de 80 países diferentes nos últimos dois anos. Além disso, esse banco de imagens foi rotulado manualmente a posição dos veículos, placas e caracteres, bem como suas classes.

Importante ressaltar que o sistema alcançou taxas de reconhecimento superiores a 95% em todos os conjuntos de dados, exceto no UFPR-ALPR (onde superou o melhor resultado anterior em 7,8%), sendo capaz de processar imagens em tempo real, mesmo quando há 4 veículos na cena. O sistema proposto superou trabalhos anteriores e sistemas comerciais em quatro conjuntos de dados públicos amplamente utilizados na literatura (Laroca; Menotti, 2020).

Em Filho (2019) é apresentado um método para localização e reconhecimento de placas automotivas baseado em duas principais etapas, a primeira consiste em localizar e extrair a região da placa da imagem de entrada utilizando operadores morfológicos. A segunda parte é feita realizando um processo de reconhecimento através da técnica de *Template Matching* por meio do coeficiente de correlação de Pearson. Os resultados são comparados entre imagens previamente cadastradas em um banco de imagens, e o sistema utiliza essa comparação para permitir ou não o acesso a um condomínio residencial por meio de uma cancela.

O experimento proposto e desenvolvido no trabalho teve um desempenho considerado satisfatório pelo autor, uma vez que atingiu uma taxa de acerto de 83% para a localização da região da placa e 97% no reconhecimento dos caracteres. O trabalho atingiu, de

maneira geral, os resultados esperados considerando que os objetivos propostos foram cumpridos, como a estruturação do banco de imagens e o desenvolvimento do algoritmo capaz de realizar o reconhecimento das placas veiculares. E por fim, comprovou-se a hipótese geral do trabalho, mostrando que é possível desenvolver um sistema de visão computacional utilizando operadores morfológicos e correlação para extrair e reconhecer placas veiculares (Filho, 2019).

No trabalho de Barreto (2018) é feito o retreinamento de um modelo de reconhecimento de placas e um modelo de reconhecimento de objetos diversos utilizando bases de dados sintéticas de placas automotivas brasileiras com variações de rotação, ruído e tamanho. A influência da utilização de placas sintéticas na acurácia de sistemas responsáveis por localizar placas reais, segmentar os caracteres e reconhecê-los foi avaliada e nos testes realizados houve um aumento da acurácia (em relação a um sistema treinado com placas reais) de três etapas: segmentação dos caracteres, reconhecimento de letras e reconhecimento dos números (2,54%, 1,09% e 2,49% respectivamente).

Um trabalho que tentou implementar um sistema embarcado foi o apresentado por Calis (2018), o objetivo do seu trabalho foi propor um sistema ALPR de baixo custo e pequenas dimensões para efetuar o controle de veículos em ambientes privados. Para isso foram utilizadas redes neurais convolucionais (CNNs ou RNCs), técnicas de *Deep Learning* para detecção da placa veicular e dos caracteres presentes nela.

O conjunto total de placas e não placas, incluindo as amostras geradas por aumento de dados, totalizaram 30183 itens. A acurácia média dos dados de treinamento foi de 99% com pequenas variações em cada *fold*. Foram necessárias 7 épocas de treinamento para atingir os resultados. A acurácia média dos testes foi de 98% com 46 variações pequenas (Calis, 2018).

Inicialmente desejava-se utilizar como hardware um Raspberry pi 3, um módulo de câmera, sensor de movimento e um servo motor para mostrar o funcionamento da cancela. No entanto na fase de testes o desempenho analisado foi pior do que o esperado. tornando impossível utilizar o Raspberry 3, principalmente pelas limitações em termos de processamento. Então ele não obteve todos os resultados esperados, principalmente pelo alto custo computacional que os algoritmos desenvolvidos exigiam. No entanto ele propôs uma correção, utilizando um método chamado de *Single Shot Detector* (SSD), que consiste em analisar apenas alguns quadros do vídeo, economizando muito processamento (Calis, 2018).

CAPÍTULO 3

Metodologia

Nesta seção, serão detalhados os procedimentos e algoritmos adotados no desenvolvimento do sistema de reconhecimento de placas de veiculares. Inicialmente, serão discutidos os métodos utilizados para o reconhecimento óptico de caracteres (OCR), abordando tanto a implementação local (embarcada) quanto a utilização de recursos em nuvem para processamento de imagens.

Em seguida, será descrito o algoritmo desenvolvido para correção de códigos identificados, visando mitigar erros comuns de OCR e aumentar a precisão do sistema. Além disso, será apresentado o método de cálculo de similaridade entre códigos, explorando a consideração da similaridade gráfica dos caracteres para uma avaliação mais abrangente. Por fim, serão discutidos os resultados obtidos com a aplicação desses métodos e algoritmos, destacando a eficácia e confiabilidade do sistema proposto para o reconhecimento de placas de veículos. Essa seção fornecerá uma visão abrangente dos processos e técnicas utilizadas, fundamentais para o funcionamento adequado do sistema e para a obtenção de resultados precisos e confiáveis.

3.1 Construção do banco de imagens

A primeira etapa do processo de construção desta monografia envolveu a preparação dos dados necessários para o treinamento do modelo utilizado na rede neural. O banco de imagens original, Laroça e Menotti (2020, p. 3), constituiu a base primária das imagens utilizadas no treinamento. No entanto, sua estrutura de dados inicial revelou-se incompatível com os requisitos do modelo a ser treinado na plataforma TensorFlow. Como resultado, foi imprescindível realizar uma conversão dos dados, utilizando scripts desenvolvidos em Python, acessíveis no repositório¹, para otimizar todo o processo.

Adicionalmente, foram introduzidas 100 novas imagens já no formato compatível. Esas novas imagens consistem em placas no modelo Mercosul, ausentes no banco de imagens original devido à data de sua criação. O conjunto de dados original compreendia

¹ <https://github.com/hiagohsantos/ALPR>

150 imagens, todas com resolução de 800x600 pixels e representando placas no modelo antigo. As 100 novas imagens adicionadas são todas do modelo Mercosul, com resolução de 1080x1920 pixels. Um exemplo das imagens do banco original e das novas imagens adicionadas pode ser visto na Figura 3.1 e Figura 3.2, respectivamente.

Figura 3.1 – Imagens do banco original.



Fonte: Autor.

Figura 3.2 – Imagens adicionadas (Modelos Mercosul).



Fonte: Autor.

Em termos gerais, o procedimento envolveu a reorganização das imagens, previamente distribuídas em uma estrutura de diretórios, para um único diretório. Também foi necessário adaptar a forma como os rótulos, representados por arquivos contendo as coordenadas das placas nas imagens, eram originalmente registrados. No banco de dados original, esses rótulos eram armazenados em arquivos de texto (*.txt*), o que não atendia aos requisitos da TensorFlow. Para suprir essa demanda, foi necessário converter esses rótulos para o formato *Extensible Markup Language* (XML), exigido pela plataforma, além de reestruturar sua organização.

Após a conclusão da conversão, foi criado um único diretório que abrigava todas as imagens utilizadas no treinamento, juntamente com seus respectivos arquivos XML. Estes últimos contêm dados essenciais, como o tamanho da imagem, o diretório, o rótulo e, crucialmente, a localização precisa da placa na imagem. No total, o conjunto de dados consiste em 250 imagens, divididas em dois grupos distintos: 230 imagens destinadas ao treinamento do modelo e 20 imagens para a validação do mesmo, cujo papel é expor a rede recém treinada a imagens nunca vistas para avaliar sua acurácia.

3.2 Treinamento do modelo de detecção de placas

A metodologia adotada para o treinamento do modelo TensorFlow segue comumente a abordagem de aprendizado por transferência. Neste processo, um modelo pré-treinado em um extenso conjunto de dados é utilizado como ponto de partida. A premissa é transferir o conhecimento obtido pelo modelo pré-treinado para uma tarefa específica, como a detecção de objetos em imagens, ou, neste caso, placas automotivas.

A rede convolucional base já contém características que são genericamente úteis para classificar imagens. No entanto, a camada final de classificação do modelo pré-treinado é específica para a tarefa de classificação original e, posteriormente, específica para o conjunto de classes em que o modelo foi treinado.

Durante o treinamento, o modelo itera sobre o conjunto de dados de treinamento, ajustando seus pesos para aprender padrões específicos relacionados à tarefa em questão. No contexto da detecção de objetos, o modelo aprende a identificar características distintivas, como bordas, texturas e formas, associadas aos objetos de interesse.

A iteração ocorre ao longo de várias épocas, onde o modelo percorre todo o conjunto de dados de treinamento. Durante cada época, os pesos do modelo são ajustados com base nas diferenças entre as previsões do modelo e os rótulos reais das imagens. Esse processo de ajuste contínuo permite que o modelo melhore sua capacidade de generalização e, assim, seja capaz de realizar detecções precisas em novas imagens não vistas durante o treinamento.

Com base na documentação da TensorFlow, foi desenvolvido um algoritmo para realizar o treinamento do modelo. Para sua execução foi utilizado um computador pessoal,

onde foi criado um ambiente virtual exclusivo para o treinamento, a fim de evitar conflitos com diferentes versões do Python e dos pacotes utilizados. Este ambiente virtual executava o *Python 3.9.18*, o pacote *tflite-model-maker 0.3.5* e outros pacotes necessários. A lista completa de pacotes e versões pode ser encontrada no arquivo *requirements.txt*, presente no repositório do projeto ².

A TensorFlow oferece a possibilidade de escolher entre diferentes arquiteturas para o treinamento. Para efeito de testes, foram realizados diversos treinamentos para avaliar o desempenho. A Tabela 3.1 representa dados extraídos da documentação que detalham todas as opções de modelos disponíveis, utilizando um Raspberry Pi 4 como referência. Cada um desses modelos apresenta características diferentes com relação a tempo de execução, acurácia e tamanho de arquivo, por conta disso o treinamento foi executado diversas vezes e vários modelos foram gerados.

Tabela 3.1 – Arquiteturas disponíveis para treinamento.

Arquitetura	Tamanho (MB)	Latência (ms)	Precisão média (AP)
EfficientDet-Lite0	4.4	37	25,69%
EfficientDet-Lite1	5.8	49	30,55%
EfficientDet-Lite2	7.2	69	33,97%
EfficientDet-Lite3	11.4	116	37,70%
EfficientDet-Lite4	19.9	260	41,96%

Fonte: Adaptado de TensorFlow (s.d., p. 1).

No processo de treinamento do modelo, uma configuração foi utilizada, na qual a rede neural foi alimentada com lotes de 10 imagens de uma só vez (*batch_size=10*). Essa configuração de agrupamento facilita o processamento das informações. Além disso, essa alimentação com os lotes de imagens foi repetida por 30 vezes (*epochs=30*), o que é conhecido como épocas. Essa configuração permite que a rede neural aprenda gradualmente com os dados, refinando suas capacidades ao longo do tempo.

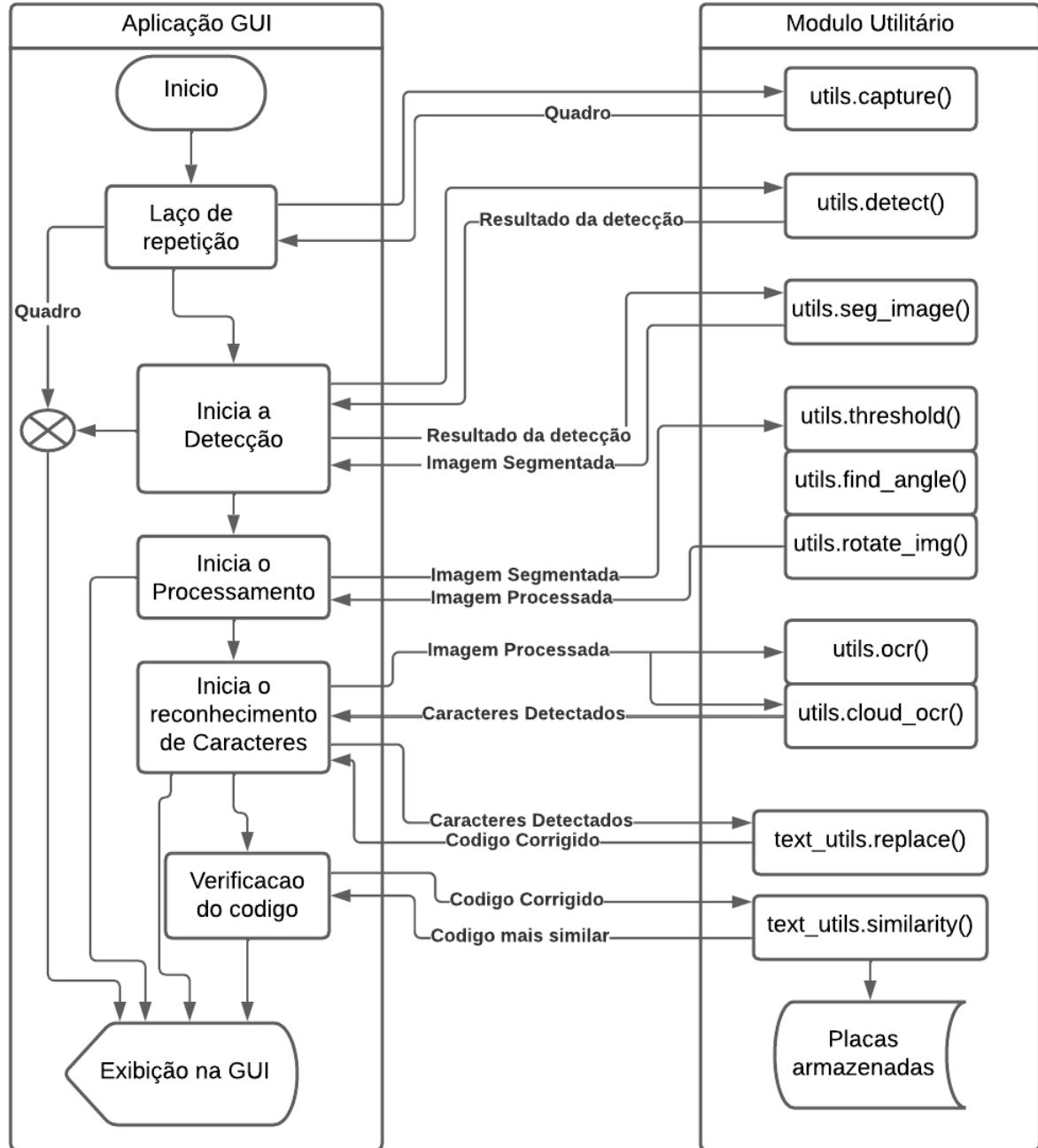
3.3 Desenvolvimento do sistema

3.3.1 Estrutura do sistema

O sistema foi estruturado de forma que a interface gráfica seja implementada em um único arquivo Python, enquanto todos os outros processamentos, tanto de imagem quanto de texto, estão organizados em métodos ou funções dentro de dois arquivos utilitários. A execução ocorre dentro do arquivo onde a interface foi implementada, de onde esses métodos são chamados de acordo com as ou ações tomadas pelo usuário. A estrutura do arquivo principal pode ser visualizada na Figura 3.3.

² <https://github.com/hiagohsantos/ALPR>

Figura 3.3 – Diagrama estrutural simplificado do sistema.



Fonte: Autor.

Na Figura 3.3, é perceptível a presença de um laço de repetição que captura constantemente quadros da câmera e, se configurado na interface, os exibe. Desta forma, conforme as configurações, esses quadros podem ser enviados para métodos dentro do próprio módulo de interface. Esses métodos são responsáveis por iniciar, de forma assíncrona para que a captura de imagem não pare, funções externas nos módulos utilitários, como a detecção de placas, processamento de imagens e processamento de texto, que serão explicadas separadamente nas próximas seções.

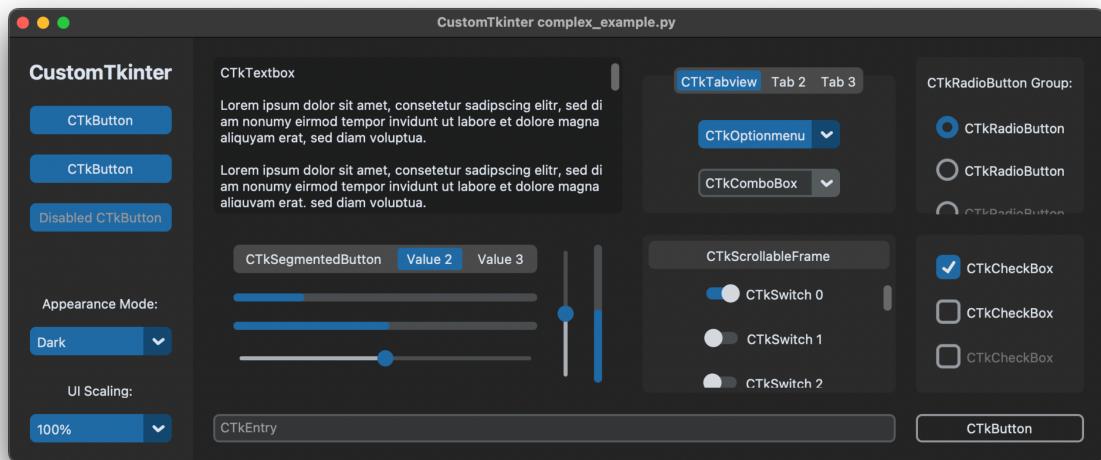
Portanto, ao manter essa separação entre os métodos que processam imagens e resul-

tados e os algoritmos de construção da interface, o código torna-se mais comprehensível e facilita também a sua manutenção.

3.3.2 Interface gráfica do sistema

Existem várias bibliotecas disponíveis em Python para o desenvolvimento de interfaces gráficas, do inglês, *Graphical User Interface* (GUI), cada uma com suas características distintas. Considerando as limitações do projeto, a prioridade recai sobre o desempenho, a fim de reservar recursos para o processamento de imagem. Com base nesse critério, a escolha recaiu sobre uma biblioteca denominada Custom Tkinter, uma versão customizada da biblioteca TKinter, disponibilizada por Schimansky (2022). Esta opção se destacou em todos os aspectos, oferecendo um excelente desempenho para interfaces relativamente modernas.

Figura 3.4 – Exemplo de uma GUI feita com a Custom TKinter.



Fonte: Schimansky (2022, p. 1).

A proposta da interface é facilitar a visualização das etapas executadas, fornecendo detalhes e exibindo imagens das operações realizadas. Além disso, ela é utilizada para efetuar diversas configurações implementadas no sistema ao longo do desenvolvimento e para adicionar placas autorizadas ao sistema, permitindo-lhe tomar decisões sobre a autorização ou não da placa encontrada.

3.3.3 Algoritmos de detecção e segmentação das placas

O método de detecção de placas de carro fundamenta-se em um modelo treinado utilizando TensorFlow Lite, mais precisamente implementado com a arquitetura EfficientDet, uma arquitetura de rede neural eficiente e escalável desenvolvida pela Google.

A função central do algoritmo, denominada *detect()*, recebe uma única imagem no formato de matriz (array numpy), aplica a TensorFlow Lite e retorna um objeto *processor.DetectionResult*. Este objeto contém os dados essenciais para as etapas subsequentes do processamento, como dados das coordenadas, se existir, da detecção da placa.

O método subsequente ao da detecção, denominado *seg_image()*, é responsável por realizar a segmentação do quadro processado. Os parâmetros do método incluem um quadro onde a detecção foi aplicada e um objeto do tipo *processor.DetectionResult*, dos quais ele extrai os dados de detecção. Com esses dois parâmetros, o método é capaz de recortar a imagem e portanto seu retorno consiste em uma imagem segmentada e informações sobre as coordenadas onde a placa foi detectada.

3.3.4 Algoritmos de processamento de imagem

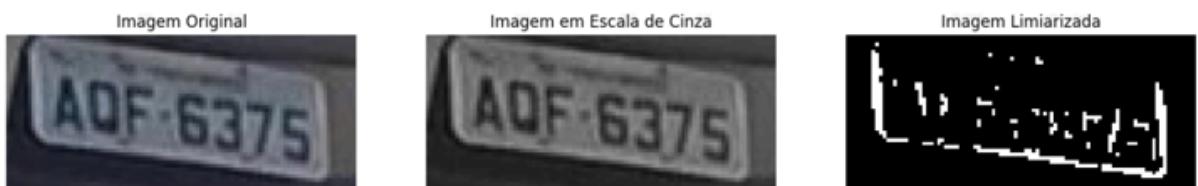
3.3.4.1 Limiarização da imagem

Após a segmentação, a imagem é encaminhada para o reconhecimento de caracteres (OCR). No entanto, em certos casos, as imagens podem não apresentar características que facilitem a detecção desses caracteres. Nessas situações, é viável aplicar algum tipo de processamento na imagem para otimizar o reconhecimento dos caracteres na próxima etapa.

O primeiro processamento realizado na imagem é a conversão para escala de cinza, visando reduzir o tamanho dos dados. Para o OCR, as cores não são relevantes, portanto, essa conversão pode reduzir significativamente o tamanho da imagem, otimizando os processos subsequentes.

O algoritmo responsável por esse pré-processamento consiste em três métodos diferentes dentro do módulo de utilitários do sistema. O primeiro deles, denominado *threshold_image()*, aplica uma técnica de limiarização na imagem. A biblioteca OpenCV oferece diversos métodos para realizar essa limiarização. O método inicialmente utilizado foi o *cv2.threshold()*, no qual os valores de limiarização são fixos e determinados pelo usuário. No entanto, os resultados obtidos com essa abordagem não foram satisfatórios em algumas imagens de teste devido as variações nas características delas, causando problemas adicionais nas etapas subsequentes, como ilustrado na Figura 3.5.

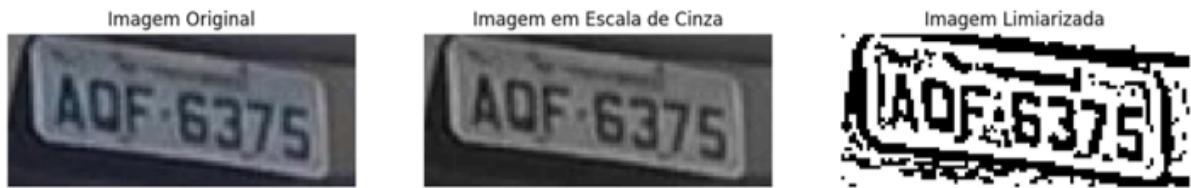
Figura 3.5 – Imagem Limiarizada com valor fixo.



Fonte: Autor.

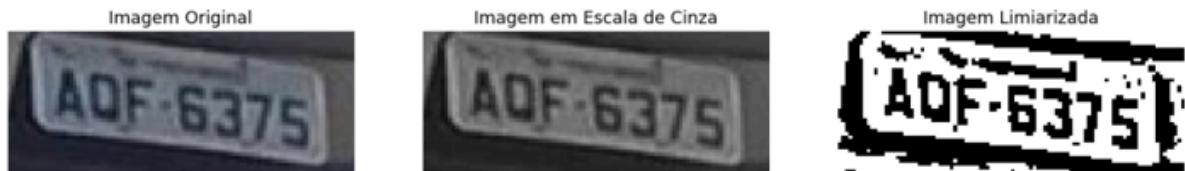
Com isso, optei por mudar a abordagem de limiarização. A OpenCV disponibiliza outras funções onde o limiar é determinado dinamicamente com base em um modelo escolhido pelo usuário (*cv2.adaptiveThreshold()*). Utilizando o modelo onde o limiar é baseado em uma soma ponderada gaussiana, conforme ilustrado na Figura 3.6, ou pelo método de média da vizinhança, como mostrado na Figura 3.7, foi obtido resultados significativamente melhores.

Figura 3.6 – Imagem limiarizada pelo método Gaussiano.



Fonte: Autor.

Figura 3.7 – Imagem limiarizada pelo método de Média.



Fonte: Autor.

Para aproveitar as implementações e testes, mantive os três modelos de limiarização, que podem ser selecionados pela interface do sistema, permitindo que o usuário escolha a opção mais adequada de acordo com o ambiente em que o sistema opera.

3.3.4.2 Angulo de inclinação da imagem

O segundo método incluído no pré-processamento é denominado *find_tilt_angle()* e é responsável por identificar a inclinação da placa na imagem segmentada. Isso é crucial, pois o OCR geralmente tem dificuldades em reconhecer caracteres quando estes estão fora da orientação horizontal.

Inicialmente, foi desenvolvido um método para encontrar o ângulo de inclinação, porém, devido à sua baixa confiabilidade, um segundo método foi implementado. No entanto, ambos foram mantidos para efeitos de comparação, permitindo que o usuário faça a escolha através da interface.

O primeiro método de detecção da inclinação utiliza uma função que identifica os contornos na imagem utilizando o método de Canny e, em seguida, aplica uma caixa delimitadora ao redor desses contornos. Esta caixa delimitadora é um retângulo com a

menor área possível que engloba o maior contorno encontrado. Ao identificar o retângulo que envolve a placa, é possível determinar a sua inclinação. A Figura 3.8 exibe um exemplo das etapas do processo.

Figura 3.8 – Processo de detecção de angulação pelo método do retângulo de área mínima.



Fonte: Autor.

Após realizar alguns testes com o método anterior, tornou-se evidente que sua confiabilidade estava comprometida devido à natureza do cálculo do retângulo de área mínima. Diante disso, iniciei a criação de um segundo método para detectar o ângulo da imagem.

Neste novo método, o primeiro passo consiste na aplicação da transformada de Canny para detectar bordas e arestas na imagem, utilizando a função `cv2.Canny()` da OpenCV. Em seguida, é aplicada a transformada de Hough para encontrar as linhas presentes na imagem. Por fim, após encontrar as linhas, é feita uma média de todos os ângulos detectados. A Figura 3.9 ilustra o processo.

Figura 3.9 – Processo de detecção de angulação pelo método de Hough.



Fonte: Autor.

Juntamente com esse método, um ultimo método é aplicado, com a detecção de angulo feita, a aplicação dele é feito multiplicando a matriz da imagem por uma matriz de rotação feita partir do angulo encontrado, o resultado já é expresso na Figura 3.8 e 3.9.

3.3.5 Algoritmo de OCR

Após o pré-processamento da imagem segmentada, o próximo passo foi tentar reconhecer os caracteres nela presentes. Para isso, utilizei dois mecanismos que operam separadamente e podem ser escolhidos pela interface do sistema. A primeira opção é totalmente embarcada, seguindo a proposta original do trabalho. No entanto, devido ao baixo desempenho, desenvolvi uma segunda opção que utiliza computação em nuvem para processar a imagem.

O método embarcado de OCR é realizado utilizando um mecanismo de código aberto chamado TesseractOCR, com o uso da biblioteca *pytesseract* em Python. Esta biblioteca oferece diversos métodos para processar imagens e extrair textos de várias maneiras diferentes. O uso do TesseractOCR permite que o sistema opere de forma totalmente embarcada, sem depender de serviços externos ou incorrer em custos adicionais. No entanto, devido às limitações de recursos disponíveis, como poder computacional da plataforma, o desempenho desse método não é muito satisfatório. Além disso, existem algumas outras limitações sobre o uso do TesseractOCR, como a baixa detecção para textos em orientações não horizontais, cujo problema foi parcialmente resolvido com o pre-processamento.

O segundo método de OCR, desenvolvido posteriormente, utiliza recursos externos para efetuar a identificação dos caracteres nas imagens segmentadas. Após o pré-processamento da imagem, ela é enviada a um serviço, também da Google, chamado Vision AI. O Vision

AI³ é um conjunto de serviços e ferramentas que utiliza tecnologias avançadas de visão computacional para análise e interpretação de conteúdo visual em imagens e vídeos. Eles disponibilizam uma *Application Programming Interface* (API) para o uso dos serviços, portanto, o sistema envia a imagem segmentada e recebe de volta dados sobre essa imagem, tudo isso por meio de requisições *Hypertext Transfer Protocol* (HTTP), utilizando o módulo Requests do Python. Para esse método, é necessário ter uma conta no serviço e que o sistema tenha acesso à internet para efetuar a comunicação.

Os métodos responsáveis por efetuar o OCR estão dentro do módulo de utilitários e são nomeados como *tesseract_ocr()* e *ocr_google_cloud()*, respectivamente. Ambos os métodos recebem uma imagem e retornam um texto contendo todos os caracteres identificados.

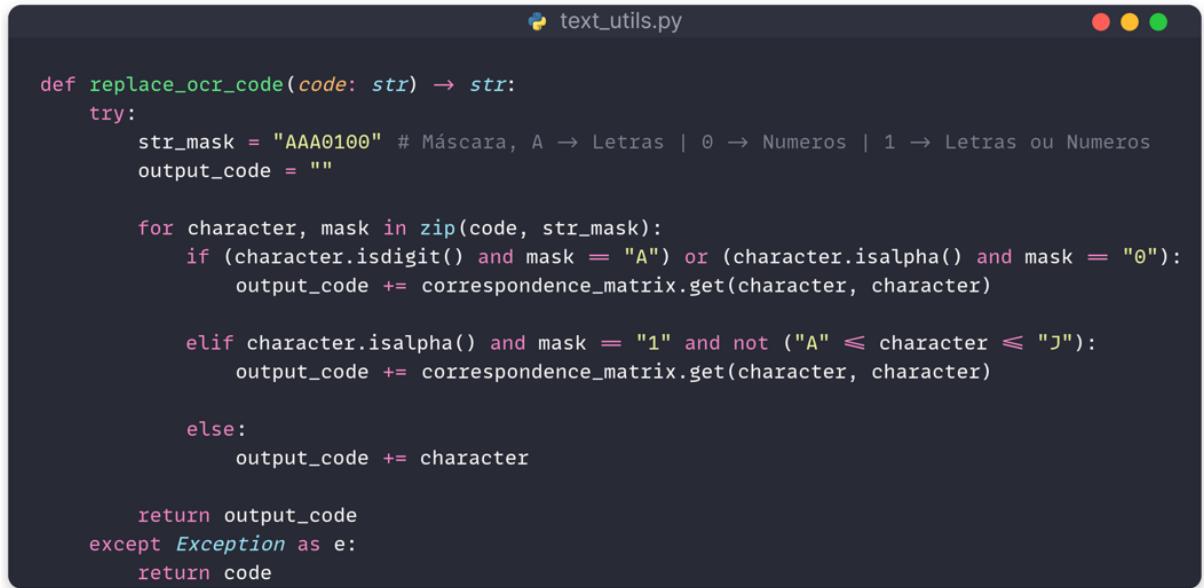
3.3.6 Algoritmo de correção de código

Com a identificação dos caracteres realizada, a próxima etapa foi processar o resultado. Muitas vezes, a etapa de OCR pode cometer erros, como a identificação incorreta de caracteres, que podem ser gerados tanto pela qualidade da imagem, pela posição dos caracteres ou por ruídos. Para mitigar esse problema, foi desenvolvido um algoritmo responsável por verificar os caracteres identificados.

O papel desse algoritmo, exibido na Figura 3.10, é verificar se os caracteres identificados podem existir na posição em que estão. Conforme mostrado na literatura, no Brasil existem atualmente dois modelos de placas vigentes, onde os três primeiros caracteres devem ser letras, e os últimos 4 podem variar dependendo do modelo, podendo haver uma letra na quarta posição no modelo Mercosul ou apenas números no modelo antigo.

³ <https://cloud.google.com/vision>

Figura 3.10 – Método responsável por corrigir caracteres nos códigos.



```

def replace_ocr_code(code: str) -> str:
    try:
        str_mask = "AAA0100" # Máscara, A → Letras | 0 → Numeros | 1 → Letras ou Numeros
        output_code = ""

        for character, mask in zip(code, str_mask):
            if (character.isdigit() and mask == "A") or (character.isalpha() and mask == "0"):
                output_code += correspondence_matrix.get(character, character)

            elif character.isalpha() and mask == "1" and not ("A" <= character <= "J"):
                output_code += correspondence_matrix.get(character, character)

            else:
                output_code += character

        return output_code
    except Exception as e:
        return code

```

Fonte: Autor.

Sabendo disso, o algoritmo percorre o texto encontrado, verificando se é possível existir o caractere na posição. Caso não seja possível, ele substitui o caractere identificado pelo correspondente mais próximo possível, de acordo com uma matriz de correspondência criada, que pode ser vista na Tabela 3.2.

Tabela 3.2 – Correspondência entre números e letras

Número	Letra	Letra	Número
O	0	0	O
I	1	1	I
Z	2	2	Z
E	3	3	E
A	4	4	A
S	5	5	S
G	6	6	G
T	7	7	T
B	8	8	B

Fonte: Autor.

Esta matriz exibe pares de letras/números que possuem similaridade gráfica e que podem ser confundidos pelo algorítimo de OCR. Por exemplo, se o algoritmo encontrar um '8' em uma posição que deveria conter apenas letras, ele substituirá o '8' por 'B'. Desse modo, o sistema poderá ganhar uma grande melhora de desempenho e acurácia. As Figuras 3.11, 3.12 e 3.13 mostram casos reais do sistema antes dessa implementação e que poderiam ter sido evitados por esse algoritmo.

Figura 3.11 – Exemplo de confusão causada pelo OCR entre o número 2 e letra Z.



Fonte: Autor.

Figura 3.12 – Exemplo de confusão causada pelo OCR entre o número 5 e letra S.



Fonte: Autor.

Figura 3.13 – Exemplo de confusão causada pelo OCR entre o número 8 e letra B.



Fonte: Autor.

Já a Figura 3.14 exibe a saída do algorítimo com o código identificado no OCR e o código já corrigido.

Figura 3.14 – Exemplos de códigos corrigidos pelo algoritmo.

String original: R102A1B	String original: A5G1K0B	String original: BRA20I2
String corrigida: RI02A18	String corrigida: ASG1K08	String corrigida: BRA2012

Fonte: Autor.

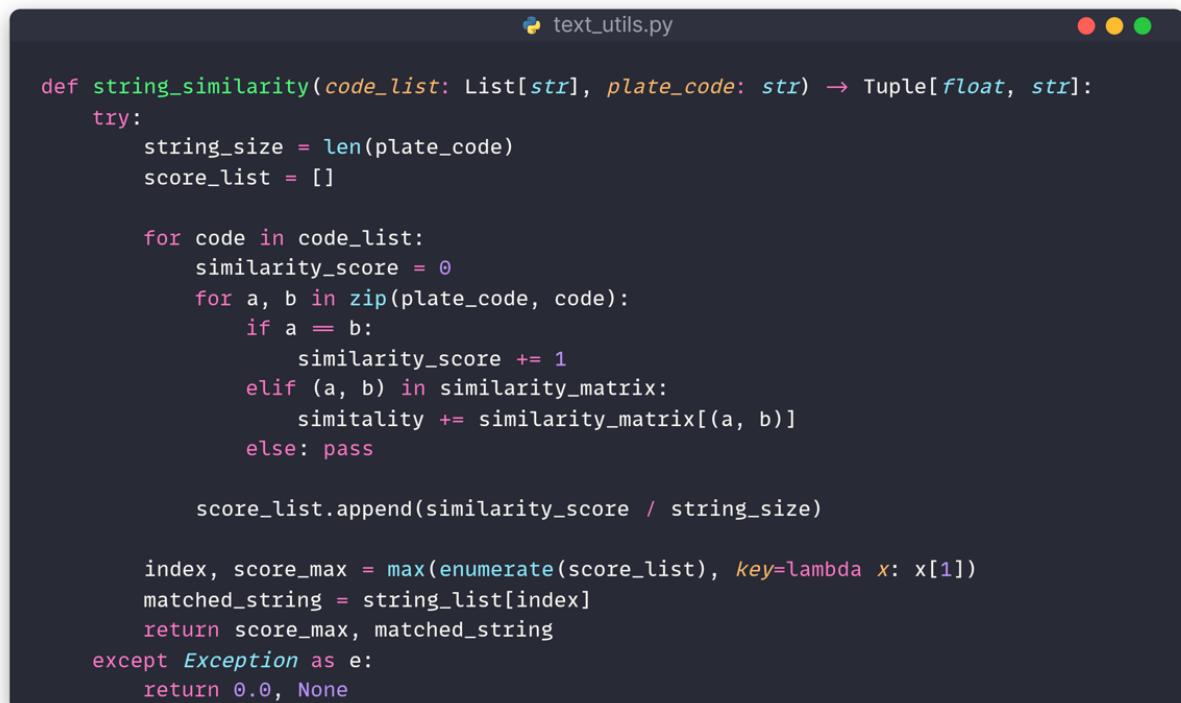
3.3.7 Cálculo de similaridade entre códigos.

A próxima etapa do sistema foi verificar se o texto obtido pertence a um código autorizado ou não, para que a decisão de autorizar ou rejeitar seja tomada. Para realizar essa verificação, um método foi criado para percorrer uma lista de placas permitidas e comparar com o resultado obtido da etapa anterior. Essa lista de placas pode ser inserida pela interface, conforme será exibido na próxima seção.

Quando executado, o método aplica sobre cada um dos códigos salvos um algoritmo que calcula a distância entre códigos ou strings. A ideia inicial era utilizar a distância de Levenshtein para verificar quão próximo o código encontrado estava dos salvos na lista. No entanto, devido às limitações desse cálculo, como não levar em consideração a similaridade gráfica dos caracteres, foi desenvolvido um novo algoritmo.

O novo algoritmo, exibido na Figura 3.15, verifica a similaridade com base não apenas na quantidade de caracteres iguais, mas também na similaridade gráfica de cada carácter. Ele percorre o código encontrado pelo OCR e um dos códigos da lista, comparando seus caracteres um de cada vez. Se os caracteres analisados forem iguais, 1 ponto é atribuído a uma variável chamada "*similarity_score*". Se não forem iguais, o código busca pelo par de caracteres em um dicionário de similaridade, representado por uma matriz de correspondência gerada, conforme explicado na seção 3.3.8 e que pode ser vista no Quadro 3.1.

Figura 3.15 – Método responsável por calcular a similaridade entre códigos.



```

text_utils.py

def string_similarity(code_list: List[str], plate_code: str) → Tuple[float, str]:
    try:
        string_size = len(plate_code)
        score_list = []

        for code in code_list:
            similarity_score = 0
            for a, b in zip(plate_code, code):
                if a == b:
                    similarity_score += 1
                elif (a, b) in similarity_matrix:
                    similarity += similarity_matrix[(a, b)]
                else: pass

            score_list.append(similarity_score / string_size)

        index, score_max = max(enumerate(score_list), key=lambda x: x[1])
        matched_string = string_list[index]
        return score_max, matched_string
    except Exception as e:
        return 0.0, None

```

Fonte: Autor.

Essa matriz de correspondência do Quadro 3.1 é uma pequena parte da matriz original, por conta das limitações do sistema, para evitar lentidões, apenas valores acima de 70% de similaridade foram levados em conta.

Ao concluir a verificação do código, o algoritmo divide a variável pelo seu tamanho, que neste caso é 7, obtendo assim um valor de similaridade bastante plausível. Esse cálculo leva em conta não apenas a quantidade de caracteres diferentes, mas também a similaridade gráfica de alguns caracteres. Dessa forma, o algoritmo consegue determinar o quão próximo o código encontrado está em relação aos códigos salvos na lista, retornando a melhor correspondência possível.

Com base nesse valor de similaridade, o sistema toma uma decisão, levando em consideração um valor mínimo de similaridade escolhido pelo usuário. Nessa escala, 100% indica que apenas códigos exatamente iguais serão aceitos. O usuário pode ajustar esse nível de confiabilidade por meio de um controle deslizante, como ilustrado na seção 4.2, Figura 4.5.

3.3.8 Similaridade gráfica entre diferentes caracteres

Durante o desenvolvimento do projeto, encontrei um desafio significativo ao calcular a similaridade entre os códigos identificados pelo OCR e os códigos autorizados armazenados. Muitas das abordagens disponíveis até então consideravam apenas a distância entre strings, negligenciando completamente a similaridade entre caracteres como **8** e **B**, **2** e **Z**, **5** e **S**, entre outros. Diante disso, foi necessário criar um algoritmo capaz de calcular a distância entre códigos levando em consideração a similaridade gráfica desses caracteres.

Para resolver essa questão, desenvolvi um pequeno script com o propósito de comparar as imagens dos caracteres e calcular sua similaridade com base na porcentagem de pixels compartilhados entre elas. Para garantir maior precisão nos resultados, utilizei a fonte original das placas de trânsito do Mercosul, conhecida como *FE-Font*. A Figura 3.16, 3.17 e 3.18 ilustram esse processo.

Figura 3.16 – Exemplo feito usando o algoritmo de calculo de similaridade (O e Q).

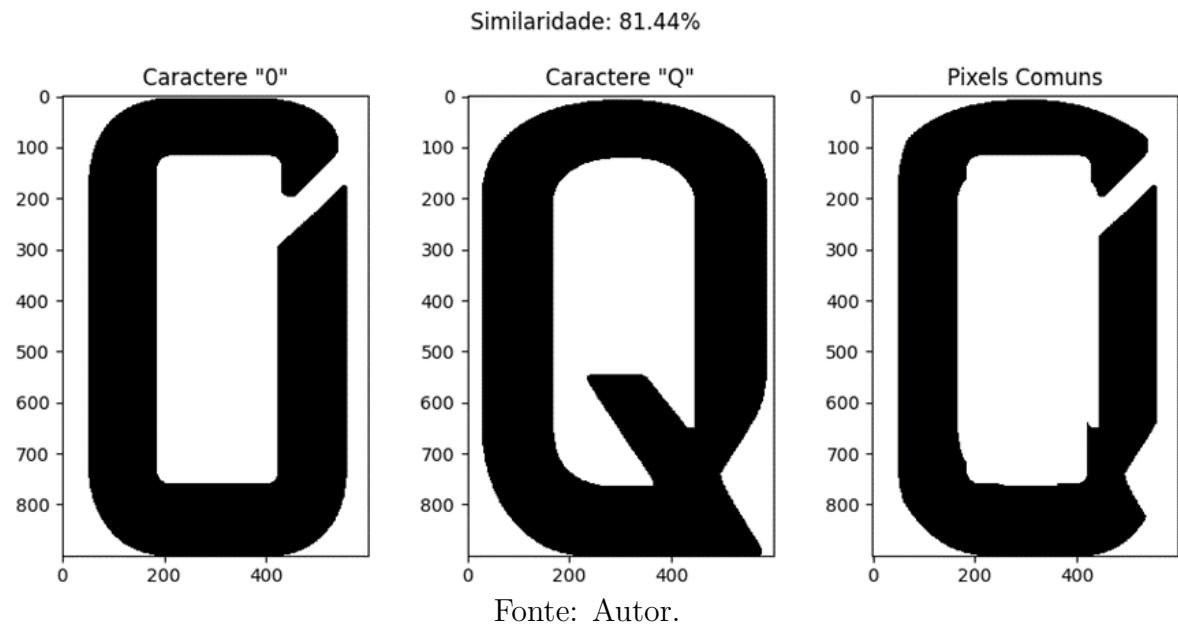


Figura 3.17 – Exemplo feito usando o algoritmo de calculo de similaridade (B e 8).

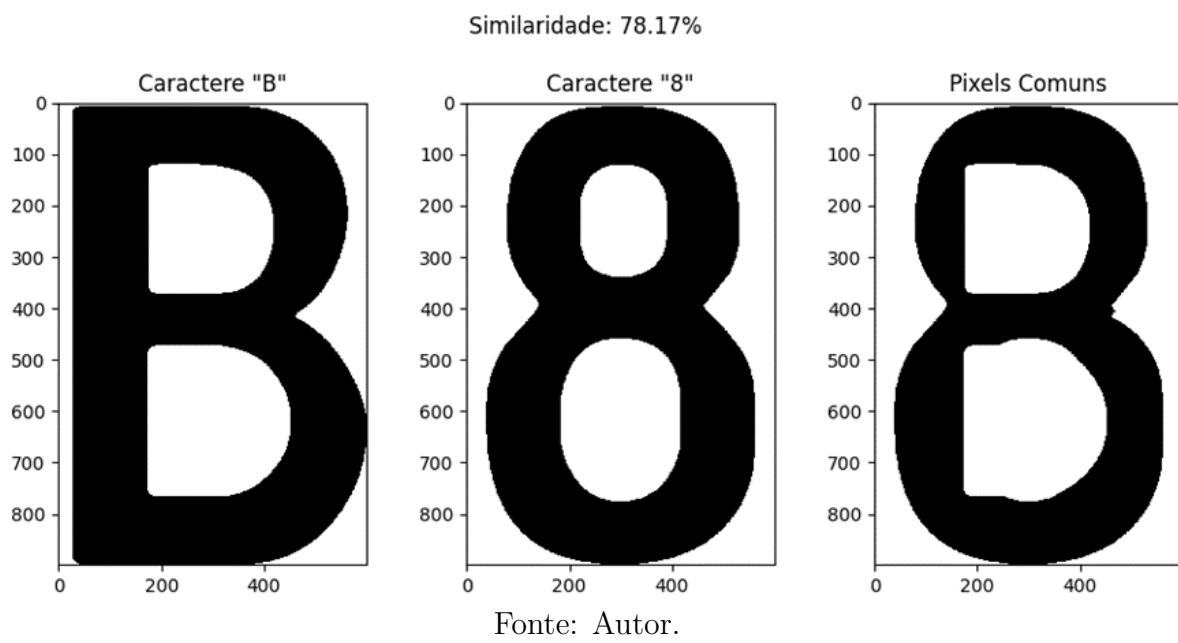
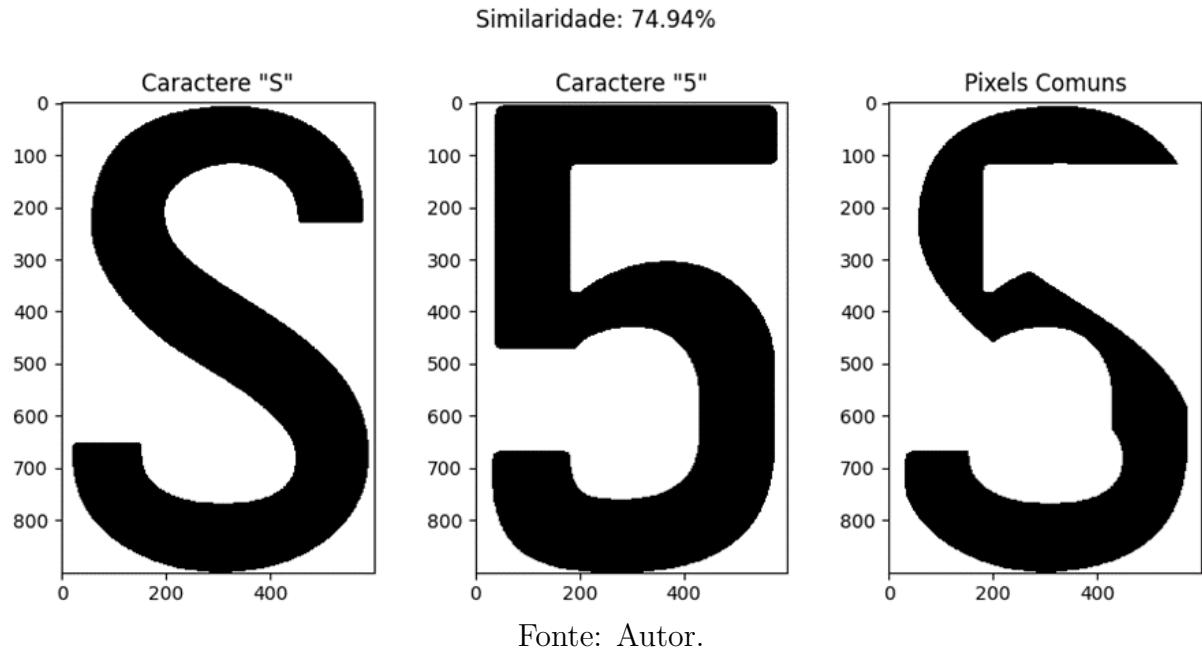


Figura 3.18 – Exemplo feito usando o algoritmo de calculo de similaridade (S e 5).



Fonte: Autor.

O cálculo foi realizado para todas as combinações possíveis (1296 combinações), utilizando um script desenvolvido para essa finalidade. A lista resultante foi normalizada, uma vez que todos os caracteres possuem pixels em comum, não havendo um verdadeiro valor zero. Em seguida, a lista foi filtrada para reduzir o impacto no desempenho do algoritmo de comparação, mantendo apenas os caracteres com semelhança acima de 70%. Isso resultou em 48 pares conforme mostrado no Quadro 3.1.

Quadro 3.1 Matriz de Similaridade.

Par de Caracteres	Similaridade	Par de Caracteres	Similaridade
B - C	0.728	G - B	0.729
B - G	0.729	G - C	0.782
B - 0	0.725	G - U	0.704
B - 8	0.703	G - 0	0.712
C - B	0.728	H - M	0.715
C - E	0.714	H - U	0.766
C - G	0.782	I - T	0.738
C - O	0.736	I - 1	0.711
C - 0	0.717	M - H	0.715
D - 0	0.748	M - N	0.79
E - C	0.714	N - M	0.79
O - C	0.736	O - Q	0.767
O - Q	0.767	O - 0	0.757
Q - O	0.767	Q - U	0.739
Q - 0	0.747	Q - 0	0.747
U - G	0.704	S - 8	0.706
U - H	0.766	T - I	0.738
U - Q	0.739	U - G	0.704
U - 0	0.726	U - H	0.766
0 - B	0.725	U - Q	0.739
0 - C	0.717	U - 0	0.726
0 - D	0.748	0 - B	0.725
0 - G	0.712	0 - C	0.717
0 - O	0.757	0 - D	0.748
0 - Q	0.747	0 - G	0.712
0 - U	0.726	0 - O	0.757
0 - 5	0.712	0 - Q	0.747
1 - I	0.711	0 - U	0.726
5 - 0	0.712	0 - 5	0.712
6 - 8	0.715	1 - I	0.711
8 - B	0.703	5 - 0	0.712
8 - S	0.706	6 - 8	0.715
8 - 6	0.715	8 - B	0.703

Fonte: Autor.

CAPÍTULO 4

Resultados

Nesta seção, uma análise abrangente do desempenho do sistema de reconhecimento de placas de veículos é apresentada, considerando diferentes configurações e métodos utilizados. Inicialmente, serão discutidos os aspectos relacionados à interface gráfica de usuário (GUI), fornecendo uma visão detalhada das funcionalidades oferecidas e sua contribuição para a usabilidade do sistema.

Em seguida, serão apresentados os resultados referentes ao desempenho do sistema, incluindo os tempos médios de processamento em cada etapa e uma análise comparativa entre diferentes configurações.

Além disso, serão abordados os resultados relacionados à confiabilidade de detecção e acurácia do sistema, considerando variações no tipo de OCR e na aplicação de correção de códigos identificados. Os dados apresentados nesta seção oferecem uma visão abrangente do desempenho do sistema e constituem uma base sólida para a avaliação de sua eficácia.

4.1 Treinamento do modelo de detecção

Utilizando a arquitetura mais eficiente disponível para treinamento da TensorFlow Lite, a *EfficientDet-Lite0*, foram realizados alguns treinamentos utilizando o banco de imagens, conforme explicados na seção 3.2. Com isso, os dados do treinamento que obteve as melhores métricas de desempenho são mostrados no Quadro 4.1.

O AP é uma métrica amplamente utilizada para avaliar a precisão de modelos de detecção de objetos. Ela calcula a precisão média ao longo de uma faixa de valores de revocação, representando a relação entre os verdadeiros positivos e os falsos positivos em uma série de limiares de confiança, como 50% e 75%. Em outras palavras, o AP mede a qualidade das detecções produzidas pelo modelo em diferentes níveis de confiança.

Por outro lado, o AR mede a capacidade do modelo em recuperar todos os objetos de uma classe específica em um conjunto de dados. Ele calcula a taxa de *recall* média para um conjunto de limiares de confiança, indicando a proporção de objetos da classe que

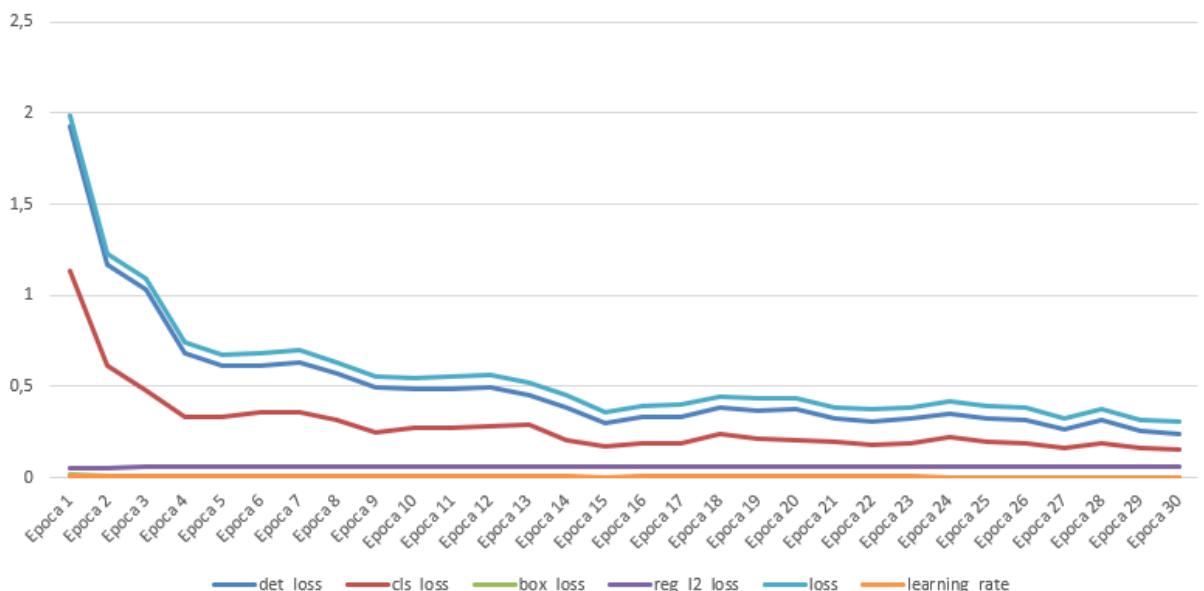
Tabela 4.1 – Métricas de Treinamento do Modelo

Métrica	Treino	Validação
<i>Average Precision (AP)</i>	0,6634	0,6342
AP ₅₀	1,0000	1,0000
AP ₇₅	0,8116	0,7883
APs	-1,0000	-1,0000
APm	0,6238	0,6023
API	0,7340	0,7046
<i>Average Recall (AR)_{max1}</i>	0,6550	0,6400
AR _{max10}	0,7450	0,6600
AR _{max100}	0,7650	0,6600
ARs	-1,0000	-1,0000
ARm	0,7538	0,6307
ARI	0,7857	0,7142
AP _{placa}	0,6634	0,6342

foram corretamente detectados em relação ao total de objetos dessa classe no conjunto de dados.

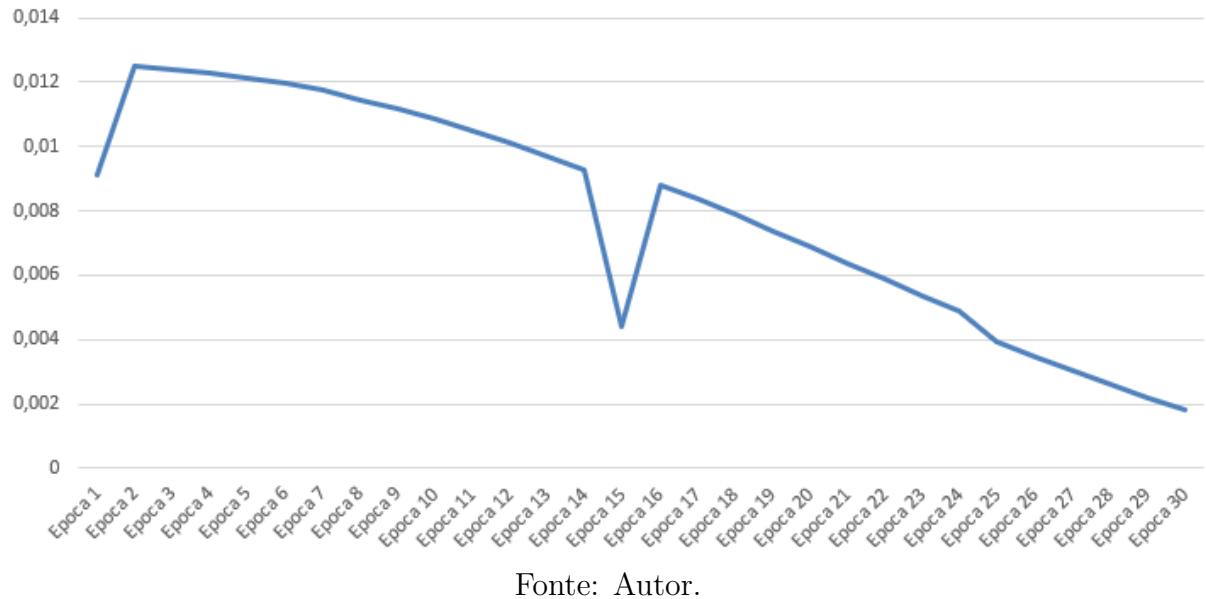
O resultado geral demonstrou que o modelo obteve uma precisão média de 66,34% no treinamento e 63,42% na validação para detectar placas automotivas em imagens. Durante o treinamento, foram coletados dados em cada uma das épocas, que representam iterações sobre o banco de imagens. Para o treinamento, foram realizadas 30 repetições no banco de imagens, alimentando a rede neural com 10 imagens por vez. A Figura 4.1 exibe as métricas do treinamento, enquanto a Figura 4.2 enfatiza a taxa de aprendizado do modelo, ambas durante as 30 épocas.

Figura 4.1 – Gráfico de desempenho do treinamento do modelo de detecção.



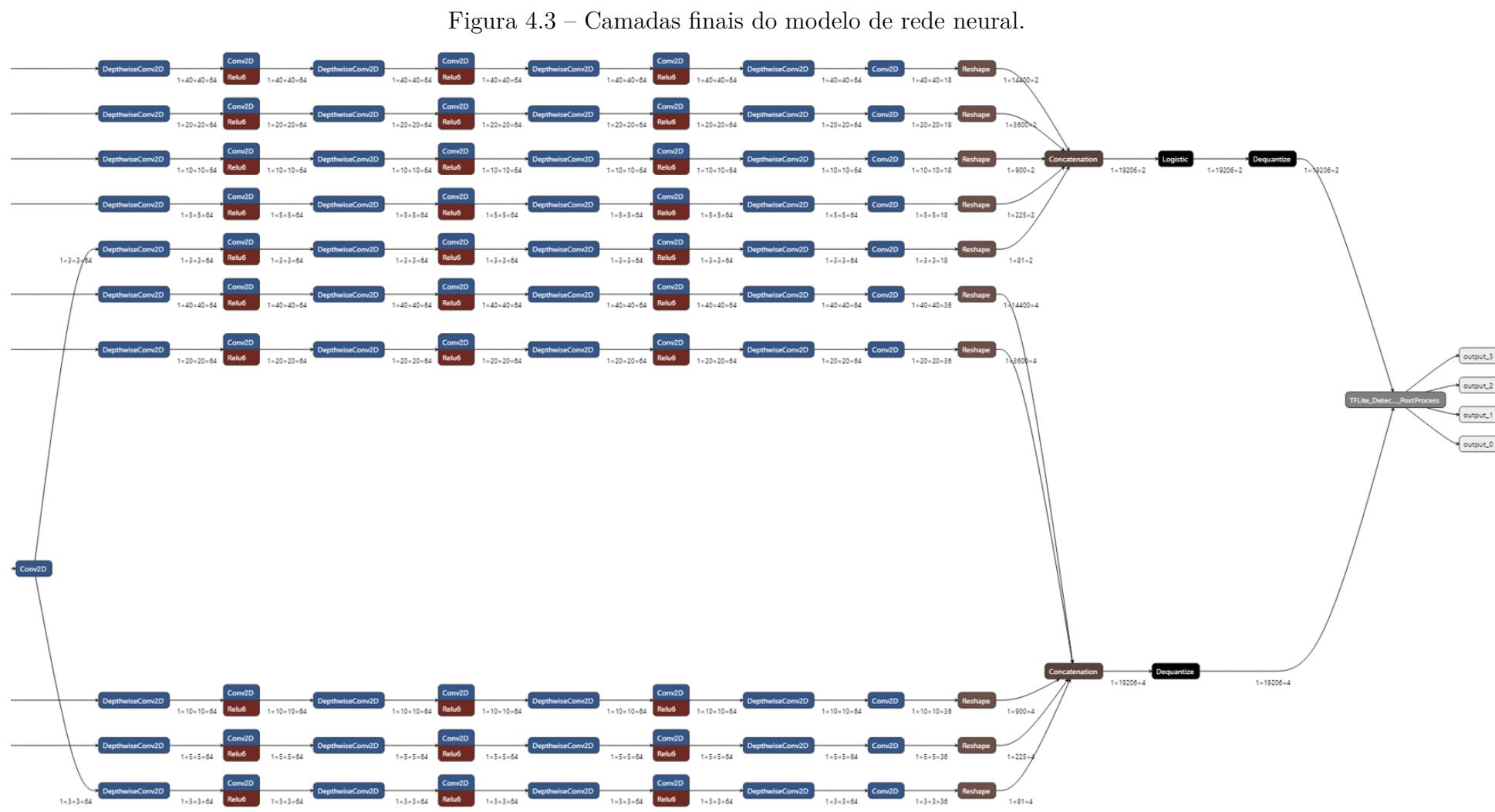
Fonte: Autor.

Figura 4.2 – Gráfico de taxa de aprendizado do modelo de detecção.



Fonte: Autor.

Também foi possível visualizar o modelo gerado do treinamento, porém devido ao tamanho do diagrama não foi possível adicionar totalmente no trabalho, no entanto a visualização das ultimas camadas pode ser visto na Figura 4.3. É possível notar o nó final se divide em 4 pontos, esses pontos retornam dados correspondentes de *numero de caixas detectadas*, a *pontuação das caixas*, as *categorias detectadas* e por fim as *coordenadas das caixas*.



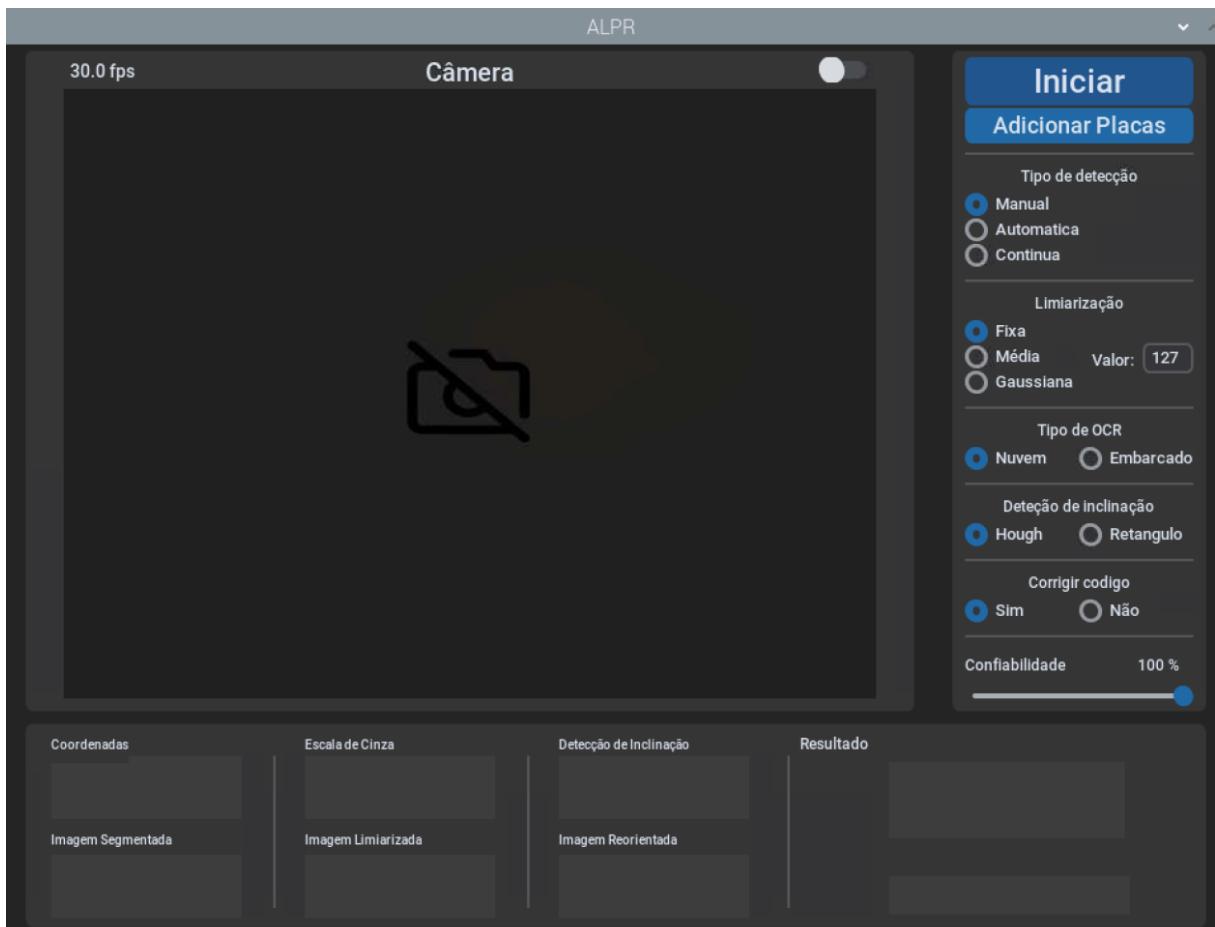
Fonte: Autor.

A visualização completa do modelo pode ser encontrada juntamente com o restante da implementação, no repositório do trabalho¹.

4.2 Interface Gráfica de Usuário (GUI)

A interface de usuário foi desenvolvida utilizando a biblioteca Python Custom TKinter². A concepção por trás da construção de uma interface gráfica é proporcionar um meio intuitivo para configurações do sistema, permitindo ao usuário a seleção das opções, detalhadas na seção 3.3, que melhor se adéquam ao ambiente de uso. Além de permitir uma visualização das etapas do processamento. A página principal da GUI do sistema está ilustrada na Figura 4.4.

Figura 4.4 – Interface principal do sistema desenvolvido.



Fonte: Autor.

A página principal é subdividida em três painéis, cada um com suas respectivas responsabilidades. O painel principal, localizado na parte superior esquerda, é destinado à exibição dos dados provenientes da câmera, em uma resolução de 800x600 pixels devido ao

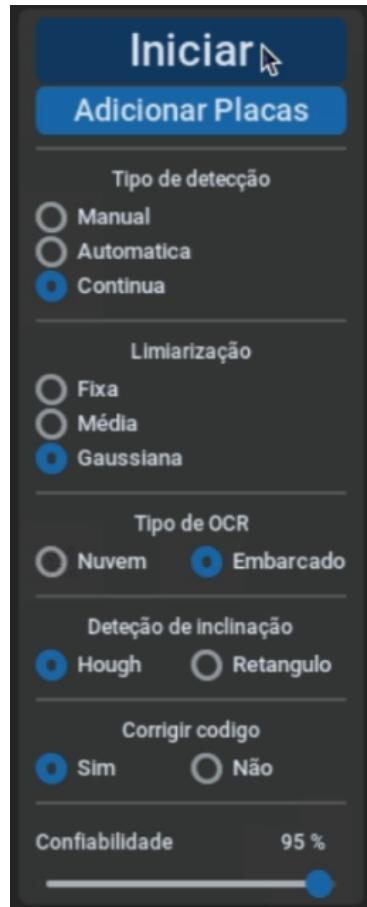
¹ <https://github.com/hiagohsantos/ALPR>

² <https://customtkinter.tomschimansky.com>

limite imposto pela câmera. Neste painel, o usuário tem a opção de visualizar ou ocultar os quadros capturados pela câmera, podendo alternar esta opção através de um controle deslizante no canto superior direito. Ademais, é apresentada uma contagem de quadros por segundo no lado oposto.

O painel superior direito, mostrado na Figura 4.5, concentra todas as configurações e ações disponíveis. Aqui, é possível selecionar o tipo de detecção, limiarização, OCR e determinar se o sistema aplicará correções nos códigos identificados, conforme detalhado na subseção 3.3.6. Além disso, o usuário pode definir o nível mínimo de confiabilidade, por meio do controle deslizante, para que o sistema autorize o reconhecimento de uma placa identificada.

Figura 4.5 – Painel lateral da interface.

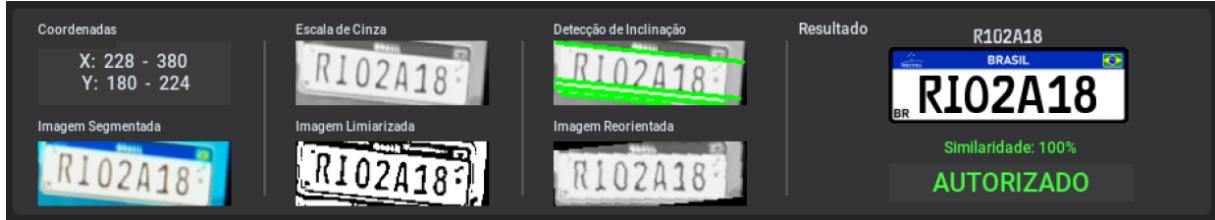


Fonte: Autor.

A parte inferior abriga um painel responsável por exibir a saída do sistema, incluindo as coordenadas de detecção, caso existam, a imagem segmentada pelo modelo de detecção, imagem limiarizada, imagem reorientada e o veredito do sistema em relação ao código da placa identificada. Um exemplo do conteúdo exibido neste painel é apresentado na Figura 4.6. Na parte direita desse painel, são exibidos o resultado do código detectado, seguido por uma imagem da placa com o código final corrigido e o veredito do sistema

com base na lista de placas.

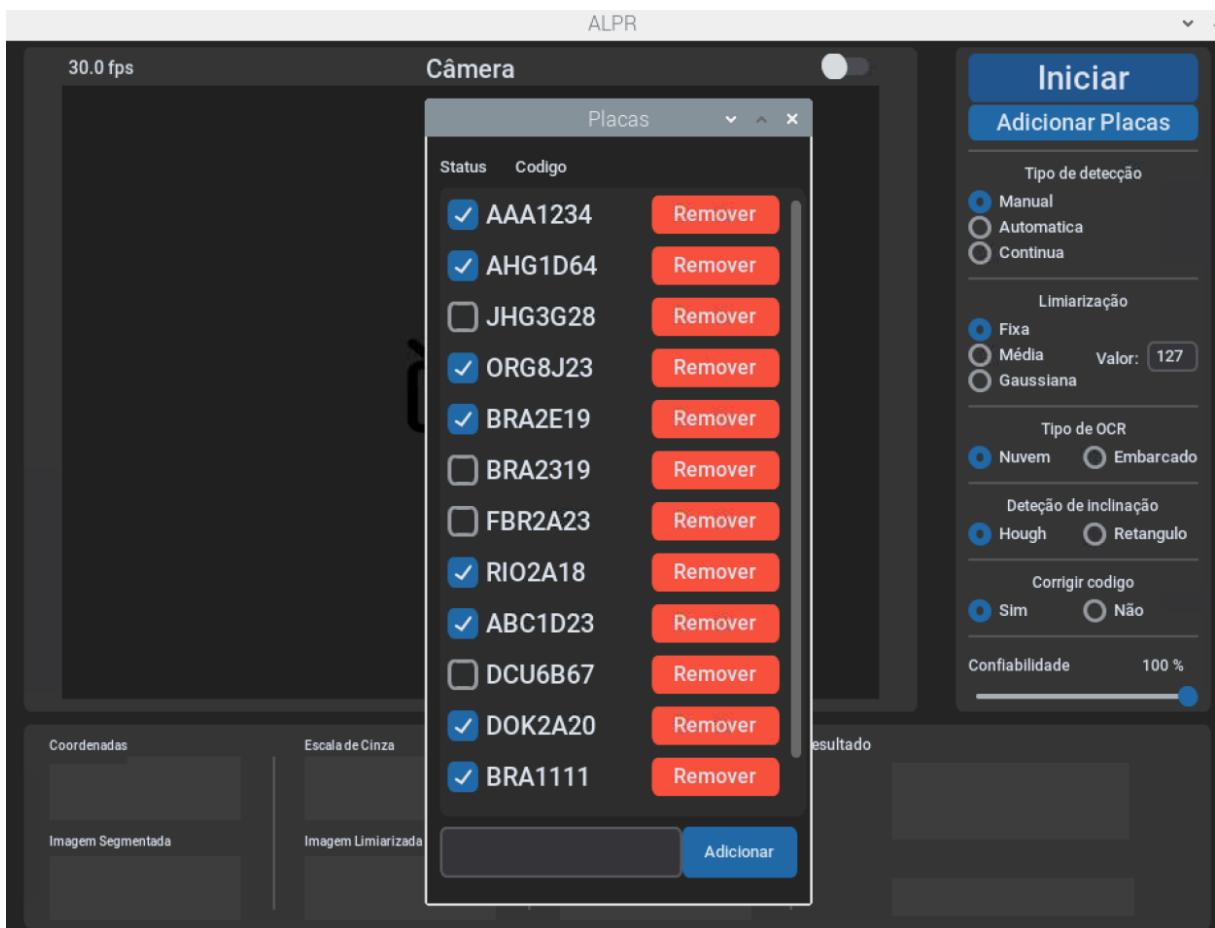
Figura 4.6 – Painel inferior da interface.



Fonte: Autor.

Existe também uma segunda página, na qual o usuário pode gerenciar as placas que serão autorizadas pelo sistema. Como ilustrado na Figura 4.7, nesta interface o usuário pode inserir ou remover códigos de placas em uma lista, tendo ainda a possibilidade de controlar sua validade por meio de uma caixa de seleção ao lado de cada código.

Figura 4.7 – Interface de adição de códigos das placas autorizadas.



Fonte: Autor.

A GUI desenvolvida desempenha um papel fundamental na usabilidade e eficácia do sistema proposto. Ao fornecer uma maneira intuitiva e visualmente atraente de interagir

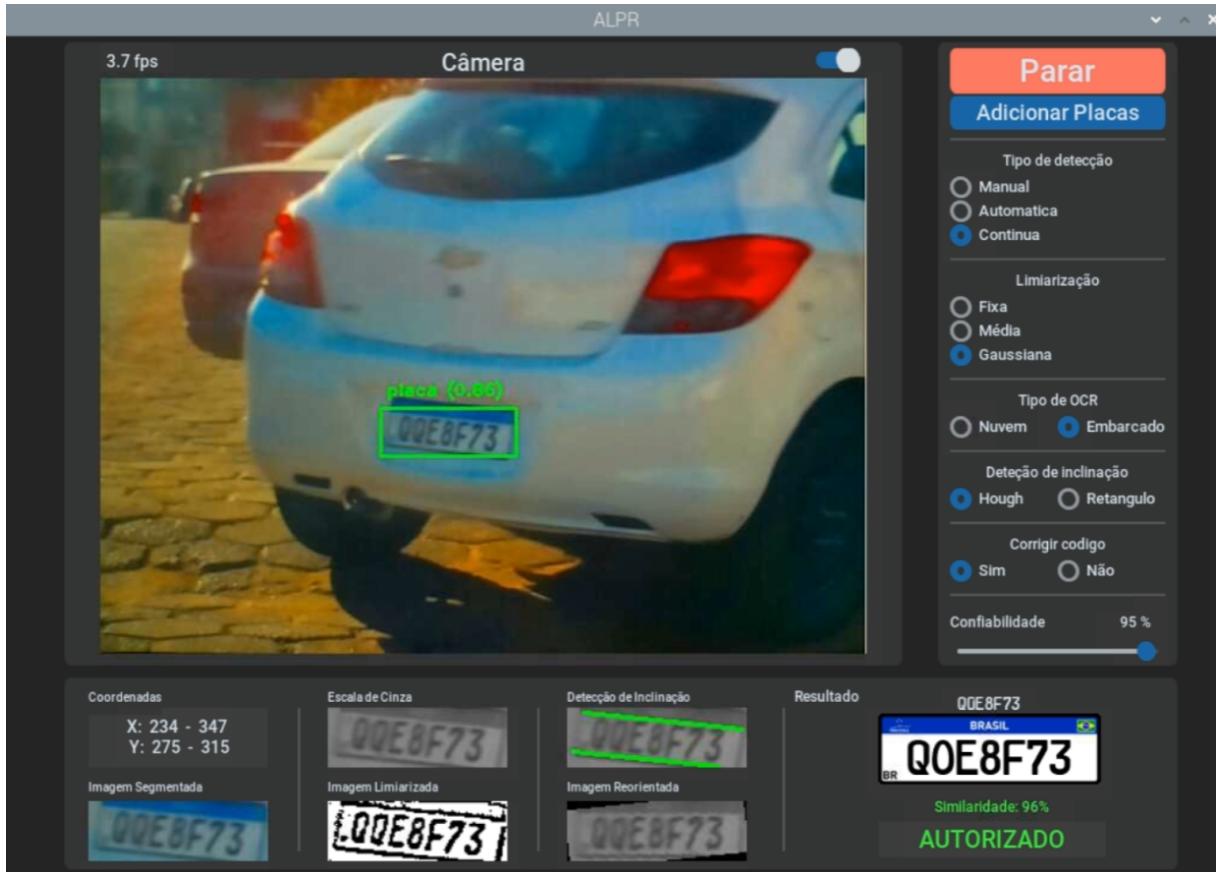
com as funcionalidades oferecidas, tornando mais acessível o processo de configuração e operação do sistema. A Figura 4.8 e Figura 4.9 mostram a interface durante o funcionamento do sistema

Figura 4.8 – Interface principal do sistema em funcionamento.



Fonte: Autor.

Figura 4.9 – Interface principal do sistema em funcionamento.



Fonte: Autor.

4.3 Desempenho do sistema

Para avaliar o desempenho do sistema, foram adicionadas ao código partes responsáveis por cronometrar o tempo gasto em cada uma das etapas principais, desde a detecção de uma placa até a obtenção do veredito final. Esse processo foi repetido várias vezes com diferentes configurações, a fim de comparar cada uma das opções disponíveis na interface. A base de imagens utilizada continha 30 imagens distintas, selecionadas do banco de imagens utilizado para treinamento e validação do modelo de detecção, conforme mostrado na seção 3.1.

Os testes realizados englobam variações no modelo de limiarização, alternando entre o Método de Média da Vizinhança e o Método Gaussiano, explicados na subseção 3.3.4.1, variações no tipo de OCR aplicado, alternando entre OCR embarcado ou OCR em nuvem, conforme mostrado na subseção 3.3.5, e, por fim, variações entre a aplicação ou não de correção no código encontrado pelo OCR, explicados na subseção 3.3.6.

A Tabela 4.2 e Tabela 4.3 exibem os tempos médios, em segundos, de processamento de cada uma das etapas para o conjunto de imagens. Os dados utilizados para o cálculo podem ser encontrados no Apêndice A, Apêndice B e Apêndice C.

Tabela 4.2 – Tempo médio das etapas do sistema (Snapshot).

	Detecção(s)	Limiarização(s)	Processamento(s)	OCR(s)
Embarcado _{Gauss}	0,1895	0,0046	0,0223	0,9634
Embarcado _{Média}	0,1987	0,0042	0,0232	0,9703
Nuvem	0,1954	0,0048	0,0217	0,7247

Fonte: Autor.

Tabela 4.3 – Tempo médio das etapas do sistema (Contínuo, máx. 5 segundos).

	Detecção(s)	Limiarização(s)	Processamento(s)	OCR(s)
Embarcado	0,2237	0,0049	0,0236	0,9748
Nuvem	0,1882	0,0045	0,0205	0,8326

Fonte: Autor.

Com base nos dados exibidos nas tabelas, é notável que apenas o tempo de processamento para o OCR é consideravelmente alterado entre as diferentes configurações do sistema, tendo um tempo menor em todos os casos para o OCR feito utilizando o serviço em nuvem.

4.4 Confiabilidade de detecção e acurácia do sistema

Com relação aos dados de confiabilidade e acurácia do sistema, também foram feitos testes com algumas variações para fins de comparação de desempenho, foram testados o OCR em Nuvem, OCR embarcado, diferentes modelos de limiarização e aplicação ou não da correção de códigos. A Tabela 4.5 exibe todos os dados de confiança da detecção feito pelo modelo e a acurácia média para cada uma das situações. Os dados completos utilizados para o cálculo da média podem ser encontrados no Apêndice A, Apêndice B e Apêndice C, ou em formato de planilha no repositório.³

Tabela 4.4 – Acurácia média para uma captura única.

	Confiança de detecção	Acurácia
Embarcado _{Média}	83,83%	80,67%
Embarcado _{Média-Corrigido}	85,77%	80,15%
Embarcado _{Gauss}	81,53%	81,83%
Embarcado _{Gauss-Corrigido}	85,43%	86,65%
Nuvem	83,57%	95,42%
Nuvem _{Corrigido}	80,90%	96,01%

Fonte: Autor.

³ <https://github.com/hiagohsantos/ALPR>

Tabela 4.5 – Acurácia média para uma captura contínua (máx. 5 segundos).

	Confiança de detecção	Acurácia
Embarcado	86,67%	93,25%
Nuvem	83,47%	99,59%

Fonte: Autor.

Com base nos valores médios extraídos, o sistema utilizando OCR embarcado obteve uma acurácia média de **82,32%**, enquanto o sistema operando com OCR em nuvem alcançou uma acurácia média de **95,71%**, ambos para o modelo de captura única, onde um único quadro do vídeo é enviado ao sistema. No caso em que o sistema opera de forma contínua, a acurácia média para o OCR embarcado foi de **93,25%**, e para o OCR em nuvem, **99,59%**.

A diferença de desempenho entre o OCR em modo embarcado e em modo nuvem foi de **13,39%** para o sistema operando com uma única captura e de **6,34%** quando operando em captura contínua. Além disso, ao comparar os modos iguais, o modelo contínuo apresentou uma melhora de **10,93%** para o OCR embarcado e uma melhoria de **3,88%** para o OCR em nuvem. Essa discrepância era esperada, considerando que o sistema em modo contínuo tem uma maior probabilidade de identificar o código correto, devido às múltiplas tentativas realizadas ao longo do tempo.

Quanto à aplicação do algoritmo de correção dos códigos, conforme explicado na seção 3.3.6, é evidente que, na maioria dos casos, houve um aumento na acurácia geral do sistema. A maior melhoria, de **4,82%**, foi observada no sistema operando em modo embarcado com limiarização gaussiana.

Tabela 4.6 – Acurácia média para uma captura contínua (máx. 5 segundos).

	Confiança de detecção	Acurácia
Embarcado	86,67%	93,25%
Nuvem	83,47%	99,59%

Fonte: Autor.

CAPÍTULO 5

Considerações Finais

Com base nos resultados obtidos a partir dos treinamentos utilizando a arquitetura EfficientDet-Lite0 e os testes realizados para avaliar o desempenho do sistema de detecção de placas automotivas em imagens, podemos concluir que o uso de técnicas avançadas de processamento de imagens e redes neurais convolucionais resultou em métricas promissoras de precisão e confiabilidade mesmo em sistemas com muitas limitações computacionais, como o Raspberry Pi 3B.

Ao analisar as métricas de treinamento do modelo, como o *Average Precision* (AP) e o *Average Recall* (AR), é observado que o modelo alcançou resultados consistentes tanto no treinamento quanto na validação e testes, indicando uma boa capacidade de generalização. Além disso, os gráficos de desempenho e taxa de aprendizado mostraram uma evolução satisfatória ao longo das épocas de treinamento, o que reforça a eficácia da abordagem adotada.

A implementação de uma *Graphical User Interface* (GUI) proporciona uma interação intuitiva e eficiente, permitindo ao usuário configurar o sistema de acordo com suas necessidades e visualizar o processo de detecção de placas em tempo real. Isso contribui para uma experiência de usuário mais amigável e facilita a utilização da aplicação em diferentes contextos.

No que diz respeito ao desempenho do sistema, os testes realizados demonstraram que o tempo de processamento para o reconhecimento óptico de caracteres (OCR) foi significativamente reduzido ao utilizar o serviço em nuvem, evidenciando a importância da escolha da infraestrutura adequada para otimizar o desempenho do sistema.

Durante o desenvolvimento do sistema, diversas limitações e desafios foram identificados, destacando-se a falta de precisão do OCR embarcado como uma das mais significativas. Esta limitação motivou a busca e a inclusão da opção de OCR em nuvem, com o objetivo de aprimorar a qualidade do sistema. Ademais, as restrições de hardware exercearam um impacto considerável no desempenho do sistema. Tanto a capacidade computacional do processador do Raspberry Pi quanto a resolução da câmera utilizada foram reconhecidas como áreas com potencial para melhoria. A baixa resolução da câmera, em

particular, comprometeu a qualidade das imagens capturadas, impactando diretamente na precisão das detecções. Além disso, um ponto de dificuldade adicional surgiu durante o treinamento do modelo de detecção. Durante a configuração do ambiente virtual para a execução dos treinamentos, muitas incompatibilidades entre pacotes foram encontradas, o que impossibilitou a conclusão dos treinamentos de forma eficiente.

Por fim, os resultados de confiabilidade e acurácia do sistema revelaram que a combinação de diferentes técnicas, como o uso de OCR embarcado ou em nuvem, modelos de limiarização e aplicação de correção de códigos, influenciou diretamente na qualidade das detecções realizadas. A melhoria na acurácia média, especialmente ao utilizar o OCR embarcado com limiarização gaussiana, destaca a eficácia das estratégias implementadas para aprimorar o processo de reconhecimento de placas automotivas.

Dessa forma, os resultados apresentados corroboram a eficiência do sistema proposto para detecção de placas automotivas em imagens, demonstrando seu potencial para aplicações práticas em diversas áreas, como segurança pública, controle de tráfego e monitoramento veicular.

5.1 Trabalhos Futuros

No que diz respeito ao desempenho do sistema, futuramente seria interessante explorar técnicas de otimização para reduzir o tempo de processamento, especialmente em relação ao reconhecimento óptico de caracteres (OCR). Isso pode envolver a investigação de algoritmos mais eficientes, a utilização de hardware mais poderoso, com uso de *Graphics Processing Unit* (GPU) ou *Tensor Processing Unit* (TPU) para acelerar o processamento, ou ainda a otimização dos algoritmos existentes para melhor aproveitamento dos recursos disponíveis.

Outra área de melhoria potencial é a expansão do conjunto de dados utilizado para treinamento e validação do modelo de detecção. Incorporar uma variedade maior de imagens, incluindo cenários de iluminação adversa, diferentes condições climáticas e placas com formatos e estilos variados, como placas de motocicletas, pode contribuir para aumentar a robustez e a capacidade de generalização do modelo.

Por último, seria altamente vantajoso conduzir uma investigação mais minuciosa sobre as diversas configurações do sistema e seu impacto na confiabilidade e precisão das detecções. Isso envolveria a realização de testes mais abrangentes, abrangendo uma gama mais ampla de parâmetros e simulando cenários mais próximos da realidade operacional de um sistema como o proposto. É importante notar que os testes realizados neste trabalho foram restritos a ambientes controlados, utilizando imagens predefinidas, o que pode limitar a generalização dos resultados para situações do mundo real.

Referências

- ABESE. **Panorama da ABESE aponta que setor de segurança eletrônica faturou R\$ 11 bilhões em 2022 - Abese.** 2023. Acesso em: 11 abr. 2023. Disponível em: <https://abese.org.br/panorama-da-abese-aponta-que-setor-de-seguranca-eletronica-faturou-r-11-bilhoes-em-2022/>.
- AFAPEMG. **História das placas para veículos.** 2019. Acesso em: 26 mar. 2023. Disponível em: <https://afapemg.com.br/consumidor/historia-das-placas-para-veiculos>.
- ARICA, N.; YARMAN-VURAL, F. An overview of character recognition focused on off-line handwriting. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 31, n. 2, p. 216–233, 2001.
- BARRETO, S. C. **Reconhecimento de Placas Veiculares utilizando Deep Learning: Análise da influência de dados sintéticos no processo de reconhecimento.** 77 p. Monografia (Monografia) — Faculdade de Tecnologia, Universidade de Brasília, Brasília, 2018.
- BATRA, P. et al. A novel memory and time-efficient alpr system based on yolov5. **Sensors**, MDPI, v. 22, n. 14, p. 5283, 2022.
- CALIS, J. O. G. **Aplicação de redes neurais convolucionais para reconhecimento automático de placas de veículos.** 68 p. Monografia — Universidade Estadual Paulista “Júlio de Mesquita Filho, São José do Rio Preto, 2018. Acesso em: 10 abr. 2023. Disponível em: <https://tcc.dcce.ibilce.unesp.br/media/2018/12/12/2adebeaedc13491d8ec7e19de050a8ae.pdf>.
- CHAUDHURI, A. et al. **Optical character recognition systems.** [S.l.]: Springer, 2017.
- CONTRAN. **Resolução nº 969/2022 - Anexos.** 2022. Acesso em: 19 mar. 2023. Disponível em: <https://www.gov.br/infraestrutura/pt-br/assuntos/transito/conteudo-contran/resolucoes/resolucao9692022anexos.pdf>.
- CROSTA, A. P. **Processamento digital de imagens de sensoriamento remoto.** [S.l.]: UNICAMP/Instituto de Geociências, 1999.
- DATAGEN. **Image Segmentation: The Basics and 5 Key Techniques.** s.d. Acesso em: 11 abr. 2023. Disponível em: <https://datagen.tech/guides/image-annotation/image-segmentation/>.

Diário do Transporte. **Denatran cria grupo técnico para decidir sobre placas padrão Mercosul.** 2018. Acesso em: 19 mar. 2023. Disponível em: <https://diariodotransporte.com.br/2018/03/27/denatran-cria-grupo-tecnico-para-decidir-sobre-placas-padrão-mercosul/>.

DOBITAOBYTE. **LPR (License Plate Recognition) Brazil - Parte 1.** 2017. Acesso em: 19 de março de 2023. Disponível em: <https://www.dobitaobyte.com.br/lpr-license-plate-recognition-brazil-parte-1/>.

EIKVIL, L. Optical character recognition. **citeseer. ist. psu. edu/142042. html**, v. 26, 1993.

EXPERT, I. **Funções de ativação: definição, características e quando usar cada uma.** 2020. Acesso em: 26 mar. 2023. Disponível em: <https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma/>.

FERNANDES, L. S. **Reconhecimento Automático De Placas Veiculares Brasileiras Incorporando Max-Margin Object Detection E Redes Neurais Convolucionais.** 72 p. Monografia (Especialização) — Departamento de Computação, Universidade Federal do Ceará, São Paulo, 2019.

FERREIRA, A. d. S. Redes neurais convolucionais profundas na detecção de plantas daninhas em lavoura de soja. 2017.

FILHO, A. W. L. O. **Reconhecimento de placas de carro para o controle de acesso a um condomínio residencial.** 46 p. Monografia (Projeto de Pesquisa) — Universidade do Estado do Amazonas, Manaus, 2019.

FILHO, O. M.; NETO, H. V. **Processamento digital de imagens.** [S.l.]: Brasport, 1999.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais - 3^aed.** [S.l.]: Editora Blucher, 2010.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning.** MIT Press, 2016. Acesso em: 25 mar. 2023. Disponível em: <http://www.deeplearningbook.org>.

GU, J. et al. Recent advances in convolutional neural networks. **Pattern Recognition**, v. 77, p. 354–377, 2018. ISSN 0031-3203. Acesso em: 19 de março de 2023. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0031320317304120>.

GUINGO, B. C.; RODRIGUES, R. J.; THOMÉ, A. C. G. Técnicas de segmentação de imagens, extração de características e reconhecimento de caracteres de placas de veículos. **Rio de Janeiro: UFRJ**, 2002.

HAYKIN, S. **Redes neurais: princípios e prática.** [S.l.]: Bookman Editora, 2001.

HOLM, T. **License Plate History.** 2022. Acessado em 26 de março de 2023. Disponível em: <https://techhistorian.com/license-plate-history/>.

IBM. **Convolutional Neural Networks.** s.d. Acesso em: 26 mar. 2023. Disponível em: <https://www.ibm.com/topics/convolutional-neural-networks>.

- INEXTECH. **ALPR/ANPR Industry News in Security, Parking, and Toll Enforcement Markets.** 2015. Acesso em: 23 mai. 2023. Disponível em: <https://www.inextechnologies.com/alpr-anpr-industry-news/>.
- JAIN, A.; DUIN, R.; MAO, J. Statistical pattern recognition: a review. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 1, p. 4–37, 2000.
- JALILI, V. Reasoning with shapes: profiting cognitive susceptibilities to infer linear mapping transformations between shapes. **arXiv preprint arXiv:1709.00158**, 2017. Acesso em: 19 mar. 2023. Disponível em: <https://arxiv.org/pdf/1709.00158.pdf>.
- JØRGENSEN, H. **Automatic license plate recognition using deep learning techniques.** Dissertação (Mestrado) — NTNU, 2017.
- LAROCA, R.; MENOTTI, D. Automatic license plate recognition: An efficient and layout-independent system based on the yolo detector. In: **Anais Estendidos do XXXIII Conference on Graphics, Patterns and Images**. Porto Alegre, RS, Brasil: SBC, 2020. p. 15–21. ISSN 0000-0000. Acesso em: 27 mai. 2023. Disponível em: https://sol.sbc.org.br/index.php/sibgrapi_estendido/article/view/12978.
- MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125–160, 2009.
- MISHRA, M. **Convolutional Neural Networks, Explained.** 2020. Acesso em: 26 mar. 2023. Disponível em: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>.
- MITHE, R.; INDALKAR, S.; DIVEKAR, N. Optical character recognition. **International journal of recent technology and engineering (IJRTE)**, Citeseer, v. 2, n. 1, p. 72–75, 2013.
- MOURA, V. S. **Reconhecimento de placas de automóveis e de seus condutores para a abertura automática de portões.** 54 p. Monografia — Universidade Federal de Campina Grande, Campina Grande, 2021.
- MRFR. **Automatic Number Plate Recognition Market Size & Forecast 2030 | ANPR Market.** 2020. Acesso em: 22 mai. 2023. Disponível em: <https://www.marketresearchfuture.com/reports/automatic-number-plate-recognition-market-7763>.
- NATALYA, K.; SERGEI, C. Software for car license plates recognition with minimal computing resources. In: **IEEE. 2022 24th International Conference on Digital Signal Processing and its Applications (DSPA).** [S.l.], 2022. p. 1–4.
- PATEL, C.; PATEL, A.; PATEL, D. Optical character recognition by open source ocr tool tesseract: A case study. **International Journal of Computer Applications**, Electronic Publication: Digital Object Identifiers (DOIs), v. 55, n. 10, p. 50–56, 2012.
- PHUNG, V. H.; RHEE, E. J. A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. **Applied Sciences**, v. 9, n. 21, 2019. ISSN 2076-3417. Acesso em: 26 mar. 2023. Disponível em: <https://www.mdpi.com/2076-3417/9/21/4500>.

- RAUBER, T. W. Redes neurais artificiais. **Universidade Federal do Espírito Santo**, v. 29, 2005.
- REHMAN, A.; MOHAMAD, D.; SULONG, G. Implicit vs explicit based script segmentation and recognition: a performance comparison on benchmark database. **Int. J. Open Problems Compt. Math.**, v. 2, n. 3, p. 352–364, 2009.
- SAKURAI, L. **Convolutional Neural Networks and MapReduce**. 2017. Acesso 26 mar. 2023. Disponível em: <https://www.sakurai.dev.br/cnn-mapreduce/>.
- SCHIMANSKY, T. **Official Documentation And Tutorial | CustomTkinter**. 2022. Acesso em: 17 mar. 2024. Disponível em: <https://customtkinter.tomschimansky.com/>.
- SILVA, S. M.; JUNG, C. R. Real-time license plate detection and recognition using deep convolutional neural networks. **Journal of Visual Communication and Image Representation**, Elsevier, v. 71, p. 102773, 2020.
- SONKA, M.; HLAVAC, V.; BOYLE, R. **Image processing, analysis, and machine vision**. [S.l.]: Cengage Learning, 2014.
- TENSORFLOW. **Object Detection with TensorFlow Lite Model Maker**. s.d. Acesso em: 17 mar. 2024. Disponível em: https://www.tensorflow.org/lite/models/modify/model_maker/object_detection.
- TMR, T. M. R. **Automatic Number Plate Recognition (ANPR) Market Outlook 2030**. 2020. Acesso em: 23 mai. 2023. Disponível em: <https://www.transparencymarketresearch.com/automatic-number-plate-recognition.html>.
- VALDEOS, M. et al. Methodology for an automatic license plate recognition system using convolutional neural networks for a peruvian case study. **IEEE Latin America Transactions**, v. 20, n. 6, p. 1032–1039, 2022.
- VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: SN. **Proceedings of the xxix conference on graphics, patterns and images**. [S.l.], 2016. v. 1, n. 4.
- VEDHAVIYASSH, D. et al. Comparative analysis of easyocr and tesseractocr for automatic license plate recognition using deep learning algorithm. In: IEEE. **2022 6th International Conference on Electronics, Communication and Aerospace Technology**. [S.l.], 2022. p. 966–971.
- WEIBEL, P. On the history and aesthetics of the digital image. **Druckrey** [2] p, Citeseer, v. 51, 1999.
- WIKIPÉDIA. **Placas de identificação de veículos no Brasil — Wikipédia, a encyclopédia livre**. 2023. Acesso em: 23 mai. 2023. Disponível em: https://pt.wikipedia.org/wiki/Placas_de_identifica%C3%A7%C3%A3o_de_ve%C3%ADculos_no_Brasil#/Distribui%C3%A7%C3%A3o_vertical_das_combina%C3%A7%C3%A3o_alfab%C3%A9ticas.
- XIE, L. et al. A new cnn-based method for multi-directional car license plate detection. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 19, n. 2, p. 507–517, 2018.

YOSINSKI, J. et al. How transferable are features in deep neural networks? **Advances in neural information processing systems**, v. 27, 2014.

Apêndices

APÊNDICE A

**Dados de captura contínua com no
máximo 5 segundos.**

Quadro A.1 Dados de desempenho do sistema configurado para OCR em Nuvem.

	Detecção (s)	Limiarização (s)	Processamento (s)	OCR (s)
1	0,1959	0,0049	0,025	0,6676
2	0,1942	0,0048	0,0202	0,9155
3	0,205	0,0071	0,029	0,4802
4	0,1615	0,0043	0,0188	1,1085
5	0,1649	0,0041	0,0172	1,1056
6	0,2368	0,0041	0,0211	0,5916
7	0,1963	0,0041	0,0197	0,4764
8	0,1818	0,004	0,0216	1,0848
9	0,1634	0,0043	0,0181	1,3142
10	0,1691	0,0048	0,0258	0,5084
11	0,1757	0,0051	0,0252	0,4941
12	0,1839	0,004	0,0174	1,1283
13	0,1738	0,0041	0,0238	1,1669
14	0,1924	0,0042	0,0143	1,0807
15	0,1971	0,0044	0,0209	1,1123
16	0,2026	0,0046	0,0208	0,6712
17	0,1634	0,0044	0,0243	0,584
18	0,1594	0,004	0,021	0,4519
19	0,1585	0,0044	0,0217	1,5533
20	0,2052	0,0041	0,017	0,6074
21	0,1696	0,0042	0,0179	0,6023
22	0,1971	0,0044	0,0146	0,607
23	0,1719	0,004	0,014	0,6193
24	0,1717	0,0045	0,0198	1,1117
25	0,2006	0,0044	0,0223	0,6456
26	0,2186	0,0048	0,0271	1,3532
27	0,2398	0,0041	0,0141	0,4862
28	0,1612	0,0044	0,0225	0,5311
29	0,2758	0,0043	0,0158	0,4505
30	0,1588	0,0047	0,0241	1,4687
Valor médio	0,1882	0,0044	0,0205	0,8326

Quadro A.2 Score de detecção e Acurácia do sistema configurado com OCR em nuvem.

	Código detectado	Código real	Score de detecção	Acurácia
1	OQL3D77	OQL3D77	0,8800	1,0000
2	DYA8I39	DYA8I39	0,8900	0,9587
3	EIF9E20	EIF9E20	0,9300	1,0000
4	DXU8E43	DXU8E43	0,8400	1,0000
5	GUY8H18	GUY8H18	0,9400	1,0000
6	ONW2I26	ONW2I26	0,8000	0,9587
7	QOJ9A83	QOJ9A83	0,9200	1,0000
8	PYZ0J72	PYZ0J72	0,4500	1,0000
9	QOJ0B11	QOJ0B11	0,7100	1,0000
10	GBG0D56	GBG0D56	0,9000	1,0000
11	QQE8F73	QQE8F73	0,8200	1,0000
12	HFF0G62	HFF0G62	0,9000	1,0000
13	KXV0A16	KXV0A16	0,8000	1,0000
14	HLE0C10	HLE0C10	0,7300	1,0000
15	RUP7I16	RUP7I16	0,8300	0,9587
16	EIW6035	EIW6035	0,8800	1,0000
17	BEE8419	BEE8419	0,8800	1,0000
18	BEM2418	BEM2418	0,8700	1,0000
19	AZU3476	AZU3476	0,9200	1,0000
20	AXI0974	AXI0974	0,9100	1,0000
21	AYB1489	AYB1489	0,8600	1,0000
22	AVI0949	AVI0949	0,8900	1,0000
23	AYU0035	AYU0035	0,8400	1,0000
24	MAK0198	MAK0198	0,8600	1,0000
25	BAP8567	BAP8567	0,8900	1,0000
26	AJG2630	AJG2630	0,8800	1,0000
27	AXW8502	AXW8502	0,7600	1,0000
28	ECZ7C77	ECZ7C77	0,8200	1,0000
29	RMO8G23	RMO8G23	0,5800	1,0000
30	RTC1D01	RTC1D01	0,8600	1,0000
Valor médio	-	-	0,8347	0,9959

Quadro A.3 Dados de desempenho do sistema configurado para OCR embarcado.

	Detecção (s)	Limiarização (s)	Processamento (s)	OCR (s)
1	0,2161	0,0047	0,0294	0,9471
2	0,2582	0,0050	0,0312	0,9090
3	0,5335	0,0116	0,0723	2,0360
4	0,2408	0,0045	0,0200	0,9285
5	0,2029	0,0045	0,0218	0,9221
6	0,1930	0,0041	0,0156	0,9269
7	0,2013	0,0046	0,0203	1,0002
8	0,2242	0,0044	0,0156	0,9411
9	0,2210	0,0044	0,0197	0,9413
10	0,2327	0,0051	0,0343	0,9469
11	0,1758	0,0048	0,0210	0,9148
12	0,2372	0,0045	0,0255	0,9496
13	0,2144	0,0046	0,0187	0,9384
14	0,2005	0,0041	0,0166	0,9371
15	0,1859	0,0048	0,0206	0,9584
16	0,2171	0,0048	0,0237	0,9309
17	0,2122	0,0043	0,0217	0,9377
18	0,2338	0,0043	0,0177	0,9438
19	0,2143	0,0045	0,0233	0,9265
20	0,1762	0,0056	0,0200	0,9382
21	0,2296	0,0049	0,0259	0,9256
22	0,1961	0,0044	0,0172	0,9819
23	0,1616	0,0045	0,0200	0,9184
24	0,2002	0,0049	0,0232	0,9327
25	0,2246	0,0046	0,0264	0,9257
26	0,2204	0,0050	0,0241	0,9394
27	0,2122	0,0048	0,0176	0,9926
28	0,1982	0,0047	0,0276	0,8894
29	0,2441	0,0051	0,0196	0,9318
30	0,2341	0,0044	0,0162	0,9321
Médio	0,22374	0,0048	0,0235	0,9748

Quadro A.4 Score de detecção e Acurácia do sistema configurado com OCR embarcado.

	Código detectado	Código real	Score de detecção	Acurácia
1	OQL3D77	OQL3D77	0,8500	1,0000
2	DYA8I39	DYA8I39	0,8600	1,0000
3	EIF9E20	EIF9E20	0,9000	1,0000
4	PXU6E43	DXU8E43	0,9100	0,8164
5	GUY8H18	GUY8H18	0,9300	1,0000
6	ONW2I26	ONW2I26	0,8000	0,9587
7	OOJ9A83	QOJ9A83	0,9100	0,9667
8	PYZ0J72	PYZ0J72	0,8300	1,0000
9	OOJ0811	QOJ0B11	0,9100	0,9243
10	GBG0D56	GBG0D56	0,9200	1,0000
11	QOE8F73	QQE8F73	0,7500	0,9667
12	HFF0G62	HFF0G62	0,9100	1,0000
13	KXV0A16	KXV0A16	0,9300	1,0000
14	HLE0C10	HLE0C10	0,9000	1,0000
15	RUP7I16	RUP7I16	0,8800	0,9587
16	FIW3N35	EIW6035	0,8200	0,5714
17	BEE8419	BEE8419	0,8900	1,0000
18	BEM2418	BEM2418	0,7600	1,0000
19	AZU3476	AZU3476	0,9100	1,0000
20	AKI0974	AXI0974	0,9300	0,8571
21	AYB1489	AYB1489	0,9000	1,0000
22	WTU7C43	AVI0949	0,8900	0,5306
23	AYU0035	AYU0035	0,8200	1,0000
24	MAK0198	MAK0198	0,8700	1,0000
25	BAP8567	BAP8567	0,9300	1,0000
26	AIG2630	AJG2630	0,8700	0,8571
27		AXW8502	0,7500	
28	ECZ7C77	ECZ7C77	0,7000	1,0000
29	AHO8G23	RMO8G23	0,9100	0,8164
30	RTC1DUL	RTC1D01	0,8600	0,8180
Valor médio			0,8667	0,9325

APÊNDICE **B**

Dados de captura única sem aplicação do algoritmo de correção.

Quadro B.1 Dados de desempenho do sistema configurado para OCR embarcado.

	Detecção (s)	Limiarização (s)	Processamento (s)	OCR (s)
1	0,1716	0,0047	0,0287	0,9380
2	0,1905	0,0047	0,0264	0,9309
3	0,1818	0,0046	0,0290	0,9359
4	0,1479	0,0043	0,0242	0,9301
5	0,1976	0,0050	0,0218	0,9426
6	0,1725	0,0043	0,0226	0,8847
7	0,1814	0,0045	0,0190	0,9418
8	0,1601	0,0045	0,0145	0,9436
9	0,1553	0,0044	0,0237	0,9496
10	0,1745	0,0049	0,0255	0,9463
11	0,1766	0,0046	0,0194	0,9191
12	0,1785	0,0042	0,0204	0,9289
13	0,1624	0,0047	0,0215	0,9331
14	0,1864	0,0042	0,0212	0,9383
15	0,2021	0,0047	0,0217	0,9296
16	0,2570	0,0046	0,0234	0,9466
17	0,1892	0,0050	0,0289	0,9447
18	0,1792	0,0043	0,0168	0,9362
19	0,3290	0,0083	0,0423	1,8071
20	0,1682	0,0045	0,0190	0,9442
21	0,1890	0,0047	0,0226	0,9282
22	0,2391	0,0043	0,0196	0,9594
23	0,1641	0,0043	0,0143	0,9415
24	0,1943	0,0044	0,0176	0,9440
25	0,1887	0,0046	0,0217	0,8926
26	0,1963	0,0047	0,0245	0,9276
27	0,1869	0,0042	0,0163	0,9351
28	0,1832	0,0047	0,0228	0,9321
29	0,1862	0,0042	0,0213	0,9370
30	0,1942	0,0042	0,0176	0,9325
Média	0,1895	0,0046	0,0223	0,9634

Quadro B.2 Score de detecção e Acurácia do sistema configurado com OCR embarcado.

	Código detectado	Código real	Score de detecção	Acurácia
1	00L3D77	OQL3D77	0,9100	0,9291
2	DYABI39	DYA8I39	0,9300	0,9576
3	EIF9EQO	EIF9E20	0,8100	0,8224
4	DXUGEKS	DXU8E43	0,9000	0,5714
5	GUY8H18	GUY8H18	0,9400	1,0000
6	ONW2126	ONW2I26	0,7900	0,9587
7	OOJ9AS3	QOJ9A83	0,9200	0,9247
8	PY20J72	PYZ0J72	0,9200	0,8571
9	Q0J0811	QOJ0B11	0,8400	0,9229
10	GRGODS6	GBG0D56	0,8400	0,6796
11	O0E8F73	QQE8F73	0,7300	0,9306
12	HFFOGO2	HFF0G62	0,8900	0,8224
13	KXVOAIG	KXV0A16	0,8600	0,7811
14	HLEOCIO	HLE0C10	0,7800	0,8893
15	RUP7II6	RUP7I16	0,9300	0,9587
16	EIWENAC	EIW6035	0,8800	0,4286
17	BEE8419	BEE8419	0,8000	1,0000
18	GEM2418	BEM2418	0,4700	0,9613
19	AZU3476	AZU3476	0,9200	1,0000
20	AKE0974	AXI0974	0,8600	0,7143
21	IVE1489	AYB1489	0,8900	0,5714
22	WAVTUTA	AVI0949	0,8400	0,1429
23	AYU0035	AYU0035	0,4700	1,0000
24	MAK0198	MAK0198	0,9000	1,0000
25	BAPBS67	BAP8567	0,9000	0,8147
26	AS62630	AJG2630	0,8800	0,7143
27		AXW8502	0,3600	
28	EGZ7C77	ECZ7C77	0,6800	0,9689
29	NHOBG23	RMO8G23	0,9100	0,7740
30	ATCIDUL	RTC1D01	0,7100	0,6339
Valor médio			0,8153	0,8183

Quadro B.3 Dados de desempenho do sistema configurado para OCR em nuvem.

	Detecção (s)	Limiarização (s)	Processamento (s)	OCR (s)
1	0,1514	0,0044	0,0245	1,3462
2	0,1927	0,0046	0,0235	0,7369
3	0,1749	0,0047	0,0248	0,5591
4	0,1617	0,0041	0,0210	1,2032
5	0,3274	0,0074	0,0350	0,6942
6	0,2146	0,0040	0,0181	0,6101
7	0,1674	0,0043	0,0192	0,4821
8	0,1944	0,0053	0,0180	0,5014
9	0,1715	0,0040	0,0161	1,1439
10	0,1885	0,0043	0,0228	0,4956
11	0,1788	0,0065	0,0245	1,1289
12	0,1971	0,0040	0,0179	0,5494
13	0,2255	0,0040	0,0151	1,1082
14	0,2297	0,0040	0,0181	1,1204
15	0,1981	0,0044	0,0195	0,4760
16	0,1907	0,0043	0,0209	0,4977
17	0,1705	0,0041	0,0185	0,4830
18	0,1568	0,0039	0,0148	0,4509
19	0,1581	0,0043	0,0213	0,4469
20	0,1610	0,0078	0,0245	0,4742
21	0,1907	0,0041	0,0188	0,5143
22	0,1694	0,0040	0,0157	0,5043
23	0,1914	0,0039	0,0138	0,8555
24	0,3187	0,0077	0,0359	1,2277
25	0,1440	0,0041	0,0187	1,0993
26	0,1801	0,0047	0,0228	0,4627
27	0,1555	0,0039	0,0184	0,4497
28	0,1851	0,0042	0,0288	0,4481
29	0,3231	0,0074	0,0347	1,2082
30	0,1930	0,0041	0,0242	0,4623
Médio	0,1954	0,0048	0,0217	0,7247

Quadro B.4 Score de detecção e Acurácia do sistema configurado com OCR em nuvem.

	Código detectado	Código real	Score de detecção	Acurácia
1	OQL3D77	OQL3D77	0,9100	1,0000
2	DYA8I39	DYA8I39	0,9100	0,9587
3	EIF9E20	EIF9E20	0,8500	1,0000
4	DXU8E43	DXU8E43	0,8200	1,0000
5	GUY8H18	GUY8H18	0,9000	1,0000
6	ONV2I26	ONW2I26	0,9100	0,8159
7	0039A83	QOJ9A83	0,8400	0,7863
8	PYZOJ72	PYZOJ72	0,7600	0,9653
9	Q0JOB11	QOJ0B11	0,5000	0,9306
10	GBGOD56	GBG0D56	0,8200	0,9653
11	QQE8F73	QQE8F73	0,5500	1,0000
12	HFFOG62	HFF0G62	0,8800	0,9653
13	KXVOA16	KXV0A16	0,9300	0,9653
14	HLEOC10	HLE0C10	0,8100	0,9653
15	RUP7I16	RUP7I16	0,8400	0,9587
16	EIN6035	EIW6035	0,6200	0,8571
17	BEE8419	BEE8419	0,9000	1,0000
18	BEM2418	BEM2418	0,8700	1,0000
19	AZU3476	AZU3476	0,9200	1,0000
20	AXI0974	AXI0974	0,9300	1,0000
21	AYB1489	AYB1489	0,9200	1,0000
22	AVT0545	AVI0949	0,8800	0,6769
23	AYU0035	AYU0035	0,9100	1,0000
24	MAK0198	MAK0198	0,9100	1,0000
25	BAP8567	BAP8567	0,9300	1,0000
26	AJG2630	AJG2630	0,9000	1,0000
27	AXM8502	AXW8502	0,8600	0,8571
28	ECZ7C77	ECZ7C77	0,8200	1,0000
29	RHO8G23	RMO8G23	0,8800	0,9593
30	RTC1D01	RTC1D01	0,5900	1,0000
Valor médio			0,8357	0,9542

APÊNDICE C

Dados de captura única com aplicação do algoritmo de correção.

Quadro C.1 Dados de desempenho do sistema configurado para OCR embarcado.

	Detecção (s)	Limiarização (s)	Processamento (s)	OCR (s)
1	0,2519	0,0459	0,1028	1,6080
2	0,1730	0,0046	0,0226	0,9240
3	0,3935	0,0085	0,0544	1,7455
4	0,2034	0,0045	0,0191	0,9336
5	0,2184	0,0041	0,0201	0,9280
6	0,2004	0,0042	0,0197	0,9382
7	0,1768	0,0043	0,0215	0,9389
8	0,1731	0,0042	0,0183	0,9217
9	0,2033	0,0041	0,0228	0,9420
10	0,1644	0,0059	0,0279	0,9423
11	0,1735	0,0047	0,0192	0,9185
12	0,2184	0,0042	0,0197	0,9281
13	0,1952	0,0043	0,0220	0,9213
14	0,1974	0,0043	0,0151	0,9382
15	0,1904	0,0045	0,0220	0,9505
16	0,1608	0,0046	0,0278	0,9360
17	0,1793	0,0052	0,0185	0,9304
18	0,1694	0,0043	0,0146	0,9371
19	0,2047	0,0043	0,0210	0,9208
20	0,1922	0,0056	0,0225	0,9358
21	0,1573	0,0048	0,0218	0,9386
22	0,1827	0,0045	0,0179	0,9478
23	0,1517	0,0042	0,0147	0,9277
24	0,1686	0,0043	0,0262	0,9236
25	0,1446	0,0044	0,0206	0,9371
26	0,1801	0,0047	0,0247	0,9388
27	0,1888	0,0042	0,0146	0,9223
28	0,1648	0,0045	0,0207	0,8799
29	0,1789	0,0043	0,0246	0,9388
30	0,1979	0,0042	0,0178	0,9182
Média	0,1918	0,0060	0,0245	0,9804

Quadro C.2 Score de detecção e Acurácia do sistema configurado com OCR embarcado.

	Código detectado	Código real	Score de detecção	Acurácia
1	OOL3D77	OQL3D77	0,9200	0,9667
2	DYA8I39	DYA8I39	0,9300	1,0000
3	EIF9E20	EIF9E20	0,8600	1,0000
4	NXU4E53	DXU8E43	0,9000	0,5714
5	GUY8H18	GUY8H18	0,9400	1,0000
6	ONW2I26	ONW2I26	0,8500	0,9587
7	OOJ9K83	QOJ9A83	0,9100	0,8239
8	PYZ0072	PYZ0J72	0,8800	0,8571
9	OOE0813	QOJ0B11	0,8900	0,6386
10	GBG0D56	GBG0D56	0,6800	1,0000
11	OOE8F73	QQE8F73	0,8900	0,9334
12	HEE0G62	HFF0G62	0,8600	0,7143
13	KXV0A16	KXV0A16	0,9000	1,0000
14	HLEUC10	HLE0C10	0,7600	0,9609
15	RUP7116	RUP7I16	0,8900	0,9587
16	SEL3AN5	EIW6035	0,8100	0,2857
17	BEE8419	BEE8419	0,9300	1,0000
18	REM2418	BEM2418	0,8100	0,8571
19	AZU3476	AZU3476	0,8900	1,0000
20	AXI0974	AXI0974	0,9300	1,0000
21	IYE1489	AYB1489	0,9300	0,7143
22		AVI0949	0,9300	
23	AYU0025	AYU0035	0,7500	0,8571
24	MAK0198	MAK0198	0,8000	1,0000
25	BAP8567	BAP8567	0,9200	1,0000
26	AJG2630	AJG2630	0,7100	1,0000
27		AXW8502	0,7100	
28	ECZ7C77	ECZ7C77	0,8100	1,0000
29	NHO8G23	RMO8G23	0,9300	0,8164
30	POICUL5	RTC1D01	0,7100	0,3490
Valor médio			0,8543	0,8665

Quadro C.3 Dados de desempenho do sistema configurado para OCR em nuvem.

	Detecção (s)	Limiarização (s)	Processamento (s)	OCR (s)
1	0,1615	0,0042	0,0196	0,9208
2	0,1598	0,0041	0,0221	1,1151
3	0,2166	0,0042	0,0259	0,4855
4	0,2123	0,0043	0,0192	1,1206
5	0,1874	0,0040	0,0183	0,5479
6	0,1652	0,0042	0,0191	0,4909
7	0,1753	0,0042	0,0236	0,7621
8	0,1714	0,0040	0,0141	0,4687
9	0,2013	0,0040	0,0179	0,4656
10	0,1621	0,0045	0,0283	0,4684
11	0,1614	0,0042	0,0165	0,4811
12	0,1786	0,0068	0,0264	0,4699
13	0,1625	0,0039	0,0141	0,4571
14	0,1703	0,0040	0,0189	0,4866
15	0,1954	0,0040	0,0179	0,4811
16	0,1782	0,0044	0,0200	0,4848
17	0,1831	0,0039	0,0210	0,7228
18	0,1591	0,0039	0,0133	0,4456
19	0,1693	0,0042	0,0181	0,4649
20	0,1809	0,0040	0,0178	1,1261
21	0,1730	0,0040	0,0200	0,4952
22	0,1927	0,0041	0,0147	0,5243
23	0,2008	0,0040	0,0199	0,4648
24	0,1635	0,0044	0,0165	0,4585
25	0,1624	0,0044	0,0250	0,4857
26	0,1881	0,0042	0,0204	0,4838
27	0,1870	0,0042	0,0149	0,7314
28	0,1676	0,0043	0,0197	0,4755
29	0,1725	0,0043	0,0236	0,4462
30	0,1913	0,0040	0,0201	0,4684
Médio	0,1784	0,0042	0,0196	0,5833

Quadro C.4 Score de detecção e Acurácia do sistema configurado com OCR em nuvem.

	Código detectado	Código real	Score de detecção	Acurácia
1	OQL3D77	OQL3D77	0,8600	1,0000
2	DYA8I39	DYA8I39	0,8900	0,9587
3	EIF9E20	EIF9E20	0,8000	1,0000
4	DXU8E43	DXU8E43	0,8400	1,0000
5	GUY8H18	GUY8H18	0,9000	1,0000
6	ONV2I26	ONW2I26	0,7700	0,8159
7	OOE9A83	QOJ9A83	0,9100	0,8239
8	PYZ0J72	PYZ0J72	0,8800	1,0000
9	QOJ0B11	QOJ0B11	0,7700	1,0000
10	GBG0D56	GBG0D56	0,7000	1,0000
11	QQE8F73	QQE8F73	0,8400	1,0000
12	HFF0G62	HFF0G62	0,9100	1,0000
13	KXV0A16	KXV0A16	0,9100	1,0000
14	HLE0C10	HLE0C10	0,6800	1,0000
15	RUP7I16	RUP7I16	0,8000	0,9587
16	COSPR13	EIW6035	0,7300	0,2857
17	BEE8419	BEE8419	0,8100	1,0000
18	BEM2418	BEM2418	0,6800	1,0000
19	AZU3476	AZU3476	0,9200	1,0000
20	AXI0974	AXI0974	0,9000	1,0000
21	AYB1489	AYB1489	0,8200	1,0000
22	AVI0949	AVI0949	0,8300	1,0000
23	AYU0035	AYU0035	0,7000	1,0000
24	MAK0198	MAK0198	0,7600	1,0000
25	BAP8567	BAP8567	0,9200	1,0000
26	AJG2630	AJG2630	0,5800	1,0000
27	AXW8502	AXW8502	0,7400	1,0000
28	ECZ7C77	ECZ7C77	0,7700	1,0000
29	RHO8G23	RMO8G23	0,7900	0,9593
30	RTC1D01	RTC1D01	0,8600	1,0000
Valor médio			0,8090	0,9601