Luiz Eduardo Pita Mercês Almeida

Estudo de caso em Reconhecimento Automático de Placas Veiculares usando linguagem de programação Python

João Pessoa

2018

Luiz Eduardo Pita Mercês Almeida

Estudo de caso em Reconhecimento Automático de Placas Veiculares usando linguagem de programação Python

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Elétrica da Universidade Federal da Paraíba como parte dos requisitos necessários para a obtenção do título de Engenheiro Eletricista.

Universidade Federal da Paraíba

Centro de Energias Alternativas e Renováveis

Curso de Graduação em Engenharia Elétrica

Orientador: Dr.José Maurício Ramos de Souza Neto

João Pessoa

2018

Catalogação na publicação
Seção de Catalogação e Classificação

A447e Almeida, Luiz Eduardo Pita Merces.
Estudo de caso em Reconhecimento Automático de Placas Veiculares usando linguagem de programação Python / Luiz Eduardo Pita Merces Almeida. - João Pessoa, 2018.
88 f. : il.

Orientação: José Maurício Ramos de Souza Neto Souza Neto.
Coorientação: Helon David de Macêdo Braz Braz.
TCC (Especialização) - UFPB/CEAR.

1. Processamento Digital de Imagens. Python. I. Souza Neto, José Maurício Ramos de Souza Neto. II. Braz, Helon David de Macêdo Braz. III. Título.

UFPB/BC

Luiz Eduardo Pita Mercês Almeida

Estudo de caso em Reconhecimento Automático de Placas Veiculares usando linguagem de programação Python

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Elétrica da Universidade Federal da Paraíba como parte dos requisitos necessários para a obtenção do título de Engenheiro Eletricista.

Trabalho aprovado. João Pessoa, 06 de novembro de 2018:

Dr.José Maurício Ramos de Souza Neto
Orientador

Dr. Helon David de Macêdo Braz
Co-Orientador

Dr. Alexsandro José Virgínio dos Santos
Avaliador

João Pessoa
2018

Este trabalho é dedicado a Mércia, Gonzaga, Rafaela, Maria do Carmo (Didi) e a Taisa, meus maiores motivadores e aqueles que sempre me cuidam.

Agradecimentos

Agradeço em primeiro lugar a Deus, sem o qual nada disso seria possível.

Em seguida a minha família, especialmente aos meus pais, que sempre me incentivaram e me apoiaram em minhas escolhas, dando-me forças para continuar mesmo com as barreiras enfrentadas, e à Taisa Del Pino, quem me apoiou nos momentos mais difíceis.

A Geraldo e Silvana Scabelo por terem me recebido e acolhido tão bem na cidade de Campinas.

Aos professores Dr. José Maurício Ramos de Souza Neto e Dr. Helon David de Macêdo Braz, pela orientação, mesmo à distância, para a realização deste trabalho.

A Guilherme Adriano Fôlego e a Filipe Costa pela parceria e suporte na Equipe de Visão Computacional do CPqD. A Norberto Alves, gerente Tecnologias de Fala, Imagem e Mobilidade da Diretoria de Suporte à Decisão e Aplicações (DSDA) e aos demais colegas da plataforma de Computação Cognitiva, pelos ensinamentos, em especial ao líder da plataforma Alan Godoy Souza Mello.

Aos professores Dr. Euler Cássio Tavares de Macêdo e Dr. Nady Rocha que me acompanharam ao longo dos anos na graduação como tutores do grupo PET Elétrica. E ao professor Dr. Alexsandro Virgínio dos Santos pelo suporte nas pesquisas em robótica e visão computacional.

E a todos aqueles que de alguma forma, direta ou indiretamente, contribuíram para a realização deste estágio.

"..., estejam unidos em amor e alcancem toda a riqueza do pleno entendimento,
a fim de conhecerem plenamente o mistério de Deus, a saber, Cristo.
Nele estão escondidos todos os tesouros da sabedoria e do conhecimento.
(Bíblia Sagrada, Colossenses 2, 2-3)

Resumo

Este trabalho aplica técnicas do processamento digital de imagens (PDI) e abordagem de redes neurais artificias (RNA) para solucionar, como estudo de caso, o problema do reconhecimento automático de placas veiculares em cenários externos de iluminação variada. Seu objetivo principal é explorar diversas técnicas de PDI durante as etapas de localização da placa, segmentação de caracteres e reconhecimento óptico de caracteres (OCR), para servir como guia para futuros trabalhos e disciplinas. Além disso, abordou alguns algoritmos de Aprendizado de Máquinas, RNA e Deep Learning para as etapas de OCR e localização de veículos, como uma forma de demonstrar a utilidade dessas ferramentas no campo da Visão Computacional. Esse trabalho focou na utilização da linguagem de programação Python, por essa ser uma das principais linguagens utilizadas nas áreas de visão computacional e aprendizado de máquinas. Para auxiliar, utilizou-se uma série de bibliotecas nas quais as principais foram o OpenCV para PDI e o TensorFlow Keras para a construção da RNA. Pensando em elaborar um guia que unificasse partes textuais teóricas com códigos de programação em um único documento, escolheu-se utilizar a ferramenta Jupyter Notebook. Para a realização dos testes, duas bases de imagem para reconhecimento automático de placas veiculares fornecidas pelas Universidades Federais de Minas Gerais e de Santa Catarina foram adquiridas. Os resultados obtidos para o estudo de caso foram inferiores aos obtidos com as técnicas mais modernas encontradas na literatura. Entretanto, foram úteis como demonstração para o guia proposto.

Palavras-chave: guia. processamento digital de imagens. python. reconhecimento automático de placas veiculares.

Abstract

This work applies digital image processing (DIP) techniques and artificial neural networks (ANN) approach to solve, as a case study, the problem of automatic license plate recognition in external scenarios of varied lighting. Its main purpose is to explore various DIP techniques during the steps of plate detection, character segmentation and optical character recognition (OCR), to serve as a guide for future work and disciplines. In addition, it approached some Machine Learning, ANN and Deep Learning algorithms for the OCR and vehicle detection steps, as a way to demonstrate the usefulness of these tools in the field of Computer Vision. This work focused on the use of the Python programming language, since it is one of the main languages used in the areas of computer vision and machine learning. In order to help, a set of libraries were used, in which the main ones were the OpenCV for DIP and the TensorFlow Keras for the ANN build. Thinking on developing a guide that unified theoretical textual parts with programming codes into a single document, the Jupyter Notebook tool was chosen. Also, the Jupyter Notebook tool was used as a way to build guides that unified theoretical textual parts and programming codes into a single document. To perform the tests, two image databases for automatic license plate recognition supplied by the Universidade Federal de Minais Gerais and Santa Catarina were acquired. The results obtained for in case study were lower than those obtained with the most modern techniques found in the literature. However, they were useful as a demonstration for the proposed guide.

Keywords: guide. digital image processing. python. automatic license plate recognition.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Nos últimos 20 anos, a evolução tecnológica tem tornado o homem cada vez mais dependente das máquinas. A internet, *smartphones* e os computadores têm revolucionado as relações humanas. Na indústria, as máquinas estão cada vez mais autônomas e conectadas entre si, nesse caso, a evolução dos protocolos de comunicação tem levado ao que se chama hoje de indústria 4.0. Na medicina, robôs têm auxiliado médicos em cirurgias. Nas cidades, surgiu o termo Cidades Inteligentes, conceito que denota o uso de informações obtidas de câmeras e diversos outros sensores espalhados pela cidade e interconectados por dados em nuvem para se extrair informações que auxiliem a Defesa Civil, a polícia, os órgãos de trânsito, a prefeitura e demais instituições nas suas atividades. São inúmeros exemplos de como a tecnologia tem ganhado espaço na rotina das pessoas, indo dos sistemas de busca de informações online como o Google Search, passando pelos assistentes virtuais e pelo universo de objetos ditos inteligentes.

Um dos principais responsáveis por essa evolução foi o aumento da capacidade de processamento dos computadores. Isso possibilitou a aplicação de técnicas e algoritmos propostos há cerca de meio século, mas que não puderam ser aplicadas devido ao alto custo computacional que necessitavam. Exemplos dessas técnicas são as redes neurais, conceito que surgiu por volta do final da década 1940 com von Neumann, e os algoritmos genéticos, propostos em 1975 por John Holland. Ambos os algoritmos são baseados em processos biológicos e fazem parte do conceito de Inteligência Artificial (IA) segundo Ballard e Brown (1982). Os autores definem IA como a área de estudo que tem por objetivo dotar os computadores com capacidade de processamento de informações similar ao dos organismos biológicos e que se iniciou com a ideia dos computadores digitais de Alan Turing em 1950 (BALLARD; BROWN, 1982).

Nesse contexto, a evolução dos equipamentos de captura de áudio e imagem e a aplicação de conceitos do Processamento Digital de Sinais (PDS) fez com que surgissem o áudio e a imagem digital. Com eles vieram os computadores capazes de lidar com entradas sensoriais. A aplicação de conhecimentos do PDS nas imagens digitais e a ideia de computadores capazes de imitar o sistema de visão humano levou à criação do Processamento Digital de Imagem (PDI) e concomitantemente da Visão Computacional.

A Visão Computacional é o estudo da extração de informações de uma imagem, ou seja, é a descrição explícita de objetos físicos a partir de imagens. Enquanto o PDI realiza o processamento das imagens através de transformações da imagem em outra imagem, a Visão Computacional é o entendimento da imagem, ou seja, se utiliza do PDI como pré-requisito para o reconhecimento, a manipulação e a tomada de decisão sobre objetos

em uma imagem (BALLARD; BROWN, 1982).

Nos últimos dez anos, o Aprendizado de Máquinas (*Machine Learning*) tem substituído o PDI como a base para a visão computacional. O Aprendizado de Máquinas consiste de uma área da Inteligência Artificial que se utiliza de técnicas da estatísticas, como a regressão, e de algoritmos inteligentes, como redes neurais artificiais, lógica *fuzzy*, e algoritmo evolutivos, para solucionar problemas. As soluções do *Machine Learning* são conhecidas como caixas pretas, pois o programador não precisa compreender qual a lógica ou função que mapeia as entradas e saídas de um sistema, bastando apenas montar um modelo e informar ele quais as entradas e quais as saídas desejadas.Desse modo, essas técnicas permitem em teoria solucionar uma infinidade de problemas sem que o programador seja especialista na área, o que facilita a difusão do seu uso em Visão Computacional. Em muitos problemas de visão percebeu-se que o ML apresentou soluções mais robustas que as obtidas com anos de estudo em PDI. Apesar das inúmeras discussões sobre a substituição do PDI pelo ML, diversos artigos científicos têm demonstrado que o uso de PDI como Engenharia de *Features*, isto é, na extração de informações e para aperfeiçoamento de imagens de entrada produzem significativas melhorias nos resultados do ML. Além disso, sistemas de ML possuem limitações como a grande quantidades de dados que essa técnica exige, o custo computacional para lidar com esses dados e, em muitas aplicações, o ML produz soluções mais lentas que aplicações de PDI.

Na Visão Computacional a técnica de Machine Learning mais utilizada são as Redes Neurais Artificial. Dentre as redes mais recorrentes destacam-se as RNN (Recurrent Neural Network) e CNN (Convolutional Neural Network). As RNN se caracterizam por utilizar informações sequenciais e ganharam apoio no campo do Processamento de Linguagem Natural, especialmente no estudo de fala por considerar a sequência de entrada dos dados, ou seja, o tempo em sua análise (LIPTON; BERKOWITZ; ELKAN, 2015; KOLEN; KREMER, 2001). Já as CNNs são as principais redes usadas, sendo base de muitos algoritmos vencedores das principais competições de Visão Computacional do mundo como as competições do Kaggle(KAGGLE, INC, 2018) e do ImageNet(DENG et al., 2009). As CNN utilizam-se de camadas convolucionais para extrair descritores da entrada, assim atuam como os filtros do PDI retirando características de imagens como forma do objeto, cor, iluminação, entre outros. Essas camadas são atualizadas durante o treinamento de modo a extrair as características mais importantes para a obtenção dos resultados (CIREŞAN et al., 2011; KRIZHEVSKY; SUTSKEVER; HINTON, 2012)

A tecnologia atual possibilita a expansão da aplicabilidade das técnicas visuais, as quais podem abranger os mais diferentes contextos. O PDI e a visão computacional estão presentes na indústria, com robôs que se utilizam de câmeras para posicionar um determinado objeto, ou em máquinas de inspeção de qualidade ou ainda no uso de câmeras térmicas para avaliar o estado de uma máquina. Na agricultura, máquinas utilizam câmeras

para contar a produção, selecionar frutas por qualidade, mapear o solo com uso de imagens de satélites. Na segurança, sistemas biométricos distinguem faces e digitais, câmeras capturam movimentos suspeitos como invasão de casas (cercas virtuais), reconhecem veículos furtados mediante a interpretação da placa. No trânsito e na mobilidade urbana, câmeras são utilizadas como radar, como detector infrações, em veículos de transporte autônomos. Na acessibilidade, câmeras são utilizadas para interpretar informações e repassá-las para deficientes visuais. Na medicina, auxiliam e aperfeiçoam as imagens médicas, como raio-x, tomografias e ressonâncias. Entre tantas outras áreas e aplicações.

Uma aplicação recorrente da visão computacional é no controle de acesso de veículos. Nesses sistemas, o algoritmo de visão precisa encontrar ou detectar uma placa veicular na imagem ou vídeo e realizar o OCR (*Optical Character Recognition*) ou reconhecimento óptico de caracteres para fazer a leitura do número da placa. Com essa informação é possível permitir o acesso ou não do veículo a uma localida, fazer controle do horário de entrada e saída do veículo e ainda contabilizar a quantidade de veículos no interior da localidade.

Outra aplicação do chamado Reconhecimento Automático de Placas Veiculares (*ALPR - Automatic License Plate Recognition ou ANPR - Automatic Number Plate Recognition*) é na utilização de câmeras de vigilância em cidades e estradas. Nas cidades podem ser utilizadas para encontrar veículos roubados, placas ilegais ou mesmo infrações de trânsito. Nas estradas além das aplicabilidades acimas são utilizadas como radar.

O ALPR, em geral, pode ser feito mediante três etapas: detecção da placa, segmentação dos caracteres e OCR. Nelas podem ser aplicados diversas técnicas do PDI, como também do aprendizado de máquinas. No PDI cita-se a detecção de bordas com filtros de aguçamento, as operações morfológicas e o casamento de *templates*. No aprendizado de máquinas, redes neurais para detectar a placa ou realizar o OCR.

Com tamanha aplicabilidade e importância nos sistemas atuais, o PDI e a visão computacional tornam-se conceitos importantes para os profissionais das áreas de computação, automação e eletrônica. No intuito de iniciar o estudo dessas áreas no curso de Engenharia Elétrica da Universidade Federal da Paraíba esse trabalho se dispõem a realizar uma introdução e contextualização do PDI e da visão computacional. Para isso, será utilizado o ALPR como um estudo de caso da aplicação dessas técnicas. Esse problema pode ser utilizado para analisar todas as etapas de um algoritmo de PDI, como também, os principais capítulos apresentado nos livros de PDI, como realce, transformações espaciais, operadores morfológicos e segmentação.

1.1 Objetivos Gerais

Esse trabalho visa explorar aplicações do processamento digital de imagem baseado em um estudo de caso com reconhecimento automático de placas veiculares de modo a fomentar conteúdo para uma futura disciplina de Processamento Digital de Imagens no curso de Engenharia Elétrica da UFPB e servir como guia para futuros trabalhos.

1.2 Objetivos Específicos

Os objetivos específicos dividi os objetivos gerais em tarefas que consistem de:

- Elaborar uma série de códigos em Jupyter Notebook na linguagem de programação Python documentados com uso do Markdown que sirvam de insumo para uma futura disciplina de PDI;
- Programar códigos em linguagem de programação Python que realize a localização e reconhecimento de placas veiculares em um banco de imagem;
- Comparar os resultados obtidos com os diferentes algoritmos, podendo incluir comparações com algoritmos abertos de terceiros;
- Introduzir a programação de redes neurais em Python, aplicando essa técnica no estudo de caso realizado;

2 Fundamentação Teórica

Este trabalho envolve conhecimentos em diferentes áreas e ferramentas. Assim, de forma resumida, visando mostrar a teoria dos conhecimentos aplicados e as ferramentas de software utilizadas dividiu-se essa fundamentação em três seções: Processamento Digital de Imagens, Redes Neurais Artificiais e Ferramentas de Software.

2.1 Processamento Digital de Imagens

Como visto, a visão computacional busca auxiliar a resolução de problemas complexos, tentando imitar a cognição humana e sua habilidade em tomar decisões de acordo com as informações contidas em uma imagem. Essa tarefa pode se dividir em dois níveis: o processamento de imagens (baixo nível) e a análise de imagens (alto nível). Esse tópico aborda somente conceitos do Processamento Digital de Imagens, que, por sua vez, trata-se do conjunto de técnicas para capturar, representar e transformar imagens com o auxílio de um computador. Ele permite extrair e identificar informações das imagens e melhorar a qualidade visual de certos aspectos estruturais, facilitando ações de alto nível como a percepção humana e a interpretação automática por meio de máquinas (PEDRINI; SCHWARTZ, 2008). Nesta seção serão apresentados alguns conceitos e técnicas da Visão Computacional, focado principalmente no baixo nível ou PDI.

2.1.1 Etapas de um sistema de visão computacional

Os sistemas de Visão Computacional comumente seguem o conjunto de etapas exemplificados a seguir: aquisição; pré-processamento; segmentação; representação e descrição; e reconhecimento e interpretação.

O processo de aquisição consiste na captura da imagem por meio de um dispositivo ou sensor, transformando-a para um formato adequado para sua manipulação. Esses dispositivos podem ser câmeras, tomógrafos médicos, satélites e *scanners*. Dentre os aspectos envolvidos nessa etapa cita-se a escolha do tipo de sensor, o formato da imagem digital, as configurações de luminosidade (tempo de abertura da lente, ISO, foco, entre outros exemplos), resolução, número de níveis de cinza ou cores da imagem digital (PEDRINI; SCHWARTZ, 2008).

A etapa de pré-processamento objetiva a melhoria da qualidade da imagem vinda da aquisição, para isso se utiliza das técnicas de atenuação de ruídos, correção de contraste e brilho, a citar a equalização de histograma. Ela é seguida pela etapa de segmentação a qual consiste do processo de separar a imagem em regiões de *pixels* similares (RUSSELL;

NORVIG, 2010). A ideia utilizada é dividir a imagem nas unidades estruturais da cena ou nas que distinguem os objetos de interesse (RUSS, 1995), separando os objetos da imagem (*foreground*) das informações de fundo (*background*).

Então as regiões de interesses destacadas na segmentação, são selecionadas com base na extração de características ou propriedades que possam ser usadas para distinguir classes de objetos. O processo de extração é denominado descrição e a estrutura que permite armazenar e manipular essa seleção é chamada de representação. Uma exemplo de técnica de descrição é o uso da Transformada de Hough para distinguir objetos facilmente parametrizados, como no caso de retas e círculos (DUDA; HART, 1972). Para o reconhecimento dessas formas geométricas, tal transformação tem se mostrado computacionalmente mais veloz que as técnicas baseadas no casamento com *templates* (NIXON; AGUADO, 2012).

A partir disso pode-se partir para a última etapa que trata do reconhecimento e da interpretação dos componentes de uma imagem. Reconhecimento ou classificação é o processo de atribuição de um identificador ou rótulo aos objetos da imagem, dada as características presentes nos seus descritores. Já a interpretação atribui um significado aos objetos reconhecidos, isto é, dado a classificação atribuir um resultado ou decisão para a situação descoberta (PERELMUTER et al., 1995). Atualmente, para a etapa de classificação é recorrente o uso de redes neurais e *deep learning*. As redes neurais convolutivas têm se mostrado uma ferramenta de alta eficácia para solucionar os problemas de reconhecimento de padrões e objetos em imagens, inclusive algumas já incorporam, além da classificação, as etapas de pré-processamento, segmentação e descrição.

2.1.2 Conceitos Básicos

O objeto de operação do Processamento Digital de Imagens são as imagens digitais (ANDREAS; ABIDI, 2008). A representação de uma imagem na forma digital é feita através de matrizes, as quais mapeiam as cores da imagem real, do mundo físico, para o digital. Os elementos dessa matriz são chamados de elementos pictóricos, elementos de imagens, pels e pixel. O Pixel (contração de Picture element) é termo mais utilizado (GONZALEZ; WOODS, 2006).

Smith (1995) afirma que o pixel representa um ponto de amostragem. Por exemplo, para uma imagem colorida, um pixel é um único ponto (x,y) na imagem que contém três amostras, uma para cada cor primária que contribui para a reprodução da imagem nesse ponto.

A representação de uma imagem em apenas uma única matriz bidimensional, cujos elementos têm seus valores variando de 0 a 255, preto ao branco respectivamente, é a representação mais utilizada para uma imagem em tons de cinza (os valores podem variar

de um L_{min} a um L_{max}). Essa é a representação das imagens monocromáticas em que os tons de cinza representam o brilho da imagem.

Algo comum no PDI é a necessidade de normalizar a escala de valores dos pixels de uma imagem. Normalização é o nome dado ao processo que altera o intervalo de valores de um conjunto de dados (DODGE; INSTITUTE, 2006), para imagens é utilizado para alterar o intervalo de valores de intensidade pixel na imagem. Isso é utilizado para patronizar um banco de imagens ou para adequar a imagem a funções que necessitam de um intervalo de valores fixos.

As imagens multi espectrais ou multibandas podem ser vistas como uma composição de imagens monocromáticas, onde cada uma delas é conhecido como banda. Assim para cada pixel há valores representativos para cada banda. Exemplificando, no caso de imagens coloridas RGB, um pixel possuíra uma valor independente para cada um dos três canais R, G e B (PEDRINI; SCHWARTZ, 2008).

Em geral, as fotografias são salvas como imagens coloridas ,multibanda de três canais, RGB. A conversão desse modelo para uma imagem monocromática pode ser feita de de duas formas: média ponderada dos canais ou extração de um único canal. Os pesos para média ponderada podem variar, inclusive podem ser iguais para cada um dos canais, mas em geral utilizam-se de características do olho humano como maior sensibilidade a frequência da cor verde para ponderar esses pesos. A conversão por extração de canal mais comum é conversão para o modelo HSI (Hue, Saturation, Intensity) e extração do canal de intensidade de brilho I (RUSS, 1995). Nesse trabalho utilizou-se apenas imagens monocromáticas convertidas a partir de imagens RGB, por esse motivo não foi explorado os diversos espaços de cores existentes e suas conversões nessa fundamentação.

2.1.3 Realce de imagens

Como visto imagens em monocromática são representações dos tons de brilho de uma imagem. Cada ponto representa uma intensidade luminosa. O contraste é definido é definido como uma medida relativa da intensidade luminosa por unidade de área, em outras palavras aumentar o contraste significa aumentar a variação existente entre os tons de cinza de uma imagem, e diminuí-lo significa reduzir essa variação. O brilho e o contraste de uma imagem podem ser modificados através das chamadas transformações em escala de cinza. Tais transformações podem ser através de funções lineares e não lineares, equalização de histograma, especificação de histograma e formas mais complexas como a hiperbolização de histograma.

Histograma é a representação gráfica dos níveis de brilho de uma imagem. É definido como um conjunto de números indicando o total de pixels na imagem que apresentam um determinado nível de cinza. Os histogramas são normalmente representados por barras que

fornecem para cada nível de cinza (eixo x) o número correspondente de pixels com esse valor na imagem (eixo y). A visualização do histograma permite obter uma indicação tanto da qualidade quanto do nível de contraste e brilho médio de uma imagem (GONZALEZ; WOODS, 2006).

As formas de realce mais simples são as aplicações de funções lineares, que consistem apenas de multiplicar a matriz que representa a imagem por um escalar e/ou somar a ela um valor constante. Da Álgebra Linear sabe-se que a soma ou multiplicação de uma matriz por um escalar é igual a soma ou multiplicação desse escalar em todos os elementos da matriz. Por isso, que esse tipo de transformação são denominadas transformação ponto a ponto, pois alteram o valor de um ponto na imagem de forma independente, isto é sem considera os pontos vizinhos a ele. Além das funções lineares, podem ser aplicadas funções não lineares como função quadrática, logarítmica, raiz quadrada, exponencial entre outras transformações ponto a ponto.

2.1.3.1 Equalização de histograma

A correção de contraste por meio de funções normalmente se dá por forma empírica. Para automatizar o processo de melhoria do contraste das imagens, utilizou-se nesse trabalho a equalização de histograma. Pedrini e Schwartz (2008) afirmam que a equalização do histograma consiste em obter uma imagem com distribuição de níveis de cinza uniformes, isto é, esses níveis irão aparecer na imagem aproximadamente com a mesma frequência. Ela realça as diferenças sutis entre níveis de cinza próximos e leva a um aumento no nível de detalhes perceptíveis.

Conforme os autores uma forma de implementar a equalização de histograma é utilizando a função de distribuição acumulada de probabilidade expressa pela equação 2.1:

$$g_k = \sum_{i=0}^{k} p_f = \sum_{i=0}^{k} \frac{n_i}{n} \tag{2.1}$$

em que n_i é o número de ocorrências de um determinado tom de cinza, n é a quantidade de pixels na imagem, p_f é a probabilidade de ocorrência de um nível de cinza e g_k é a distribuição acumulada de probabilidades, k representa o k-ésimo nível de cinza no intervalo de 0 a $L-1$, em que L é a quantidade de níveis existentes, por exemplo considera-se $L = 256$ nas imagens cujos elementos são representados por um *byte*.

Para que se possa utilizar essa função para a equalização de histograma é preciso normalizar os níveis de cinza da imagem no intervalo de 0 a 1. Após isso calcula-se a função de distribuição acumulada e mapeia os valores dos pixels da imagem original com base nos valores da função de distribuição acumulada.

Outra forma de equalização de histograma é a chamada especificação de histograma. Nesse tipo de ajuste de realce, um histograma é equalizado com base em outro. Assim,

pode-se copiar a distribuição da iluminação de uma imagem para outra. Umas das métricas para se efetuar esse tipo de equalização é ao invés de utilizar a função de distribuição acumulada da imagem que se quer equalizar, utiliza-se a da imagem de referencia para mapear os valores da nova imagem (PEDRINI; SCHWARTZ, 2008).

2.1.4 Filtros Espaciais

Filtros espaciais são transformações em imagem que operam sobre o valor de um pixel levando em consideração os valores dos pixels vizinhos. Esses por essa característica de utilizar a vizinhança do pixel no cálculo de seu novo valor esses filtro também são conhecidos como máscaras, templates e janelas. Esse termo se deve ao fato desses filtros serem matrizes que atribuem pesos ao pixel e a seus vizinhos. Essa máscara desliza sobre a imagem a que se aplica o filtro através da operação de convolução. Uma imagem filtrada é gerada á medida que o centro do filtro percorre cada pixel na imagem de entrada (GONZALEZ; WOODS, 2006). A seguir serão abordados as principais classes desses filtros.

2.1.4.1 Filtros de suavização

Os filtros de suavização operam de forma similar aos filtros de frequência passa-baixas. Eles são empregados na remoção de ruídos de alta frequência espacial, isto é variações bruscas de intensidade de um pixel para o outo. Além disso, por contar do efeito de borramento provocado pela atenuação das brutas transições de intensidade, esse filtro são utilizados para homogeneizar regiões e em situações cujo efeito de borramento é desejável. Esses filtros apesar de serem bons na remoção de ruídos, normalmente alteram a forma das bordas de uma imagem, devido ao efeito de borramento. Bordas são regiões de transição abrupta de intensidade que simbolização as dimensões entre os objetos da imagem (WATT; POLICARPO, 1998).

Os filtros de suavização mais comuns são os filtros de média, mediana e gaussiano. O filtro de média diz que o novo valor do pixel central da mascara é igual a média dos seus vizinhos. De forma similar o filtro gaussiano aplicam ao pixel central a média ponderada dos pixels vizinhos em que os pesos atribuídos a cada elemento da vizinhança pertencem a uma distribuição gaussiana centrada no pixel central, de tal modo que quando mais distante for o vinho do pixel ventral menor será o peso atribuído a ele. O filtro de mediana é um filtro de suavização não linear que ordena os pixel central e seus vizinhos e extrai o valor da mediana. Outro filtro bastante utilizado é o filtro bilateral utiliza-se de dois filtros gaussianos um para a distancia espacial entre o pixel central e seu vizinho e outro para a diferença de intensidade do pixel central e seus vizinhos, com isso ele passa a considerar regiões de intensidade de brilho próxima, ganhando a característica d remoção de ruídos mantendo o formato das bordas (GOYAL; BIJALWAN; CHOWDHURY, 2012).

2.1.4.2 Filtros de aguçamento

De forma oposta aos filtros de suavização tem-se os filtros de aguçamento que operam de forma similar aos filtros passa-altas do domínio da frequência, visto que eles acentuam as variações de intensidade entre os *pixels* adjacentes. Normalmente se utilização do conceito de gradiente de funções bidimensionais para buscar regiões de transição bruscas de intensidade, como as bordas de um objeto.

Entre os principais gradientes utilizados tem-se o gradiente de Prewit e de Sobel. Por serem gradiente esses filtros costumam ser utilizados em suas formas unidimensionais, isto é, buscam transições em apenas um dos eixos x ou y. Essas formas podem ser combinadas por meio de uma transformação polar gerando os filtros de magnitude e fase pra Sobel e Prewitt. Outro filtro bastante utilizado é o filtro laplaciano. Esse filtro aproveita da segunda derivada da imagem para extrair as regiões de transição. Apesar do conceito por trás desses filtros serem complexos as matrizes que os implementam não o são (PEDRINI; SCHWARTZ, 2008).

Um filtro de aguçamento que não causa o efeito de remoção das regiões homogêneas e portanto não se insere no contexto dos detectores de borda é o filtro chamado máscara de nitidez ou unsharp mask. Ela consiste da subtração de uma imagem com sua imagem borrada e servem para aumentar a nitidez da imagem (GONZALEZ; WOODS, 2006).

2.1.4.3 Template Matching

A técnica definida como Template matching é uma técnica usada para achar as instâncias de uma máscara T dentro de uma imagem I. Basicamente, o método realiza uma convolução da máscara sobre toda a imagem e retorna a posição onde existe a menor diferença e o valor de tal diferença. É importante ressaltar que, se a máscara possuir a mesma dimensão da imagem, o resultado da abordagem é semelhante ao cálculo da diferença ponto a ponto entre as imagens.

Para estimar a diferença entre a máscara e a imagem de teste durante a convolução, existem diversas métricas (MAHALAKSHMI; MUTHAIAH; SWAMINATHAN, 2012). A técnica mais comum é a correlação e suas variações e a diferença quadrática média.

2.1.5 Segmentação

A segmentação de imagens é o processo de identificação de pixels com características similares através de rótulos ou etiquetas. Os pixel rotulados com uma mesma etiqueta representam um objeto. Em outras palavras a segmentação é responsável por distinguir os objetos de uma imagem do fundo ou distinguir um objeto do outro. (CHIPANA; IANO, 2010).

Saldanha e Freitas (2009) dividem as técnicas de segmentação em três grupos: segmentação no espaço de atributos, detecção de descontinuidades e detecção de similaridades. A primeira consiste na seleção de elementos com atributos semelhante, como por exemplo o valor de intensidade dos pixels. A segunda procura na imagem regiões de descontinuidade que podem representar a transição entre o objeto e o fundo. A terceira parte de uma região no interior de um objeto e busca regiões com mesma características ao redor dessa região.

Neste trabalho serão abordados exemplos de segmentação de cada uma das três técnicas apresentadas. Para a segmentação no espaço de atributos será explicado a limiarização e algumas de suas variações. Para a detecção de descontinuidade foi apresentado os filtros de aguçamento e será exposto o detector de bordas Canny e a Transformada de Hough para linhas. Finalmente, para a segmentação por regiões será demonstrado a técnica clássica de Watershed.

2.1.5.1 Limiarização

Esta é a forma mais simples e uma das mais importante de segmentar uma imagem. Para essa abordagem , o objeto é considerado uma região de pixels que tenham em comum uma faixa de intensidades. Assim, a limiarização utiliza somente a intensidade dos pixels para distinguir as regiões (SALDANHA; FREITAS, 2009).

A limiarização pode ser feita a partir da determinação de um limiar, em que os pixels acima daquele limiar vão possuir uma valor de intensidade e os abaixo outro valor. Ela pode acontecer de em 2 ou mais nível, sendo a limiarização em dois níveis a mais utilizada. Na limiarização em dois níveis os pixels abaixo do limiar recebem o valor de intensidade zero e os pixels acima recebem o valor de limiar um ou o máximo valor possível (Exemplo, por pixels do representados por 1 bytes de variáveis inteiras positivas (uint8) o valor máximo para o pixel é 255). Já as limiarizações com dois ou mais níveis definirão mais valores de limiares para segmentar a imagem em mais de um grupo ou nível.

Quando um único limiar é aplicado sobre toda a imagem diz-se que esse limiar é global. Ao passo que, quando definimos um limiar para uma parte ou janela da imagem, tem-se uma limiarização local ou adaptativa.

A forma automática de otimização limiar mais comum é a utilização do método de Otsu (OTSU, 1979). Esse método ver a limiarização como um problema teórico de estatística cujo objetivo é minimizar o erro médio devido a atribuição de pixels para dois ou mais níveis. Na literatura esse problema é solucionado por meio da regra de decisão de Bayes que se baseia na função densidade de probabilidade e na probabilidade que cada classe ou nível ocorra em uma determinada aplicação. Esse método é de difícil aplicação, assim Otsu (1979) propôs um método mais simples cuja principal ideia foi a maximização da variância entre as classes. De modo geral o método Otsu aplica todos os possíveis

valores de limiar sobre o histograma de uma imagem e determina como ótimo aquele cujas as classes formadas possuem a maior variância entre elas considerando apenas os valores de intensidade dos pixels (GONZALEZ; WOODS, 2006).

2.1.5.2 Detectores de bordas

Bordas são as variações abruptas de intensidade que distinguem um objeto do fundo. Esses pontos de mudanças de intensidade são encontrados por meio do cálculo da primeira e da segunda derivada de uma imagem. O gradiente é a ferramenta utilizada para calcular a derivada e encontrar a intensidade e a direção da borda na posição (x, y) de uma imagem. Dada uma imagem f, o gradiente (∇f) é dado pela equação 2.2.

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.2}$$

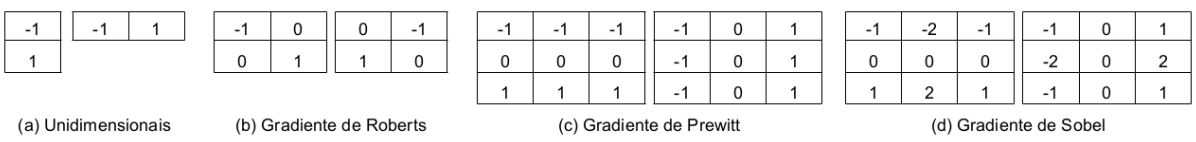
A magnitude do gradiente é o valor da taxa de variação na direção do vetor gradiente. A direção ou ângulo ou fase do vetor gradiente é medido em relação a eixo x, a direção da borda é ortogonal à direção do gradiente. Respectivamente, magnitude e fase são calculados como descrito nas equações 2.3 e 2.4.

$$mag(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{2.3}$$

$$\alpha(x, y) = tg^{-1} \begin{bmatrix} g_y \\ g_x \end{bmatrix} \tag{2.4}$$

Os operadores de gradiente são métodos de aproximação para as derivadas de uma imagem discreta. Eles definem fatores como vizinha a serem utilizada e pesos para os elementos da vizinhança. O principio básico para esse cálculo é a subtração de um pixel com o pixel anterior para cada um dos eixos. Essas subtrações podem ser representadas por máscara unidimensionais. O gradiente cruzado de Roberts foi uma das primeiras tentativas de utilizar uma máscara 2D para dar preferencias as transições na diagonal. Após isso, diversas máscaras foram propostas, entre as mais comuns as máscaras de Prewitt e Sobel, utilizadas nesse projeto (GONZALEZ; WOODS, 2006). As máscaras unidimensionais, de Roberts, Prewitt e Sobel podem ser visualizadas na Figura 1 .

Figura 1 – Máscaras dos principais detectores de bordas clássicos.



Em 1986, Canny (1986) propôs um método mais complexo de detecção de bordas. Sua abordagem se baseou em três objetivos: baixa taxa de erros, os pontos de bordas devem estar bem próximos das bordas verdadeira e uma única resposta para os cantos de bordas encontrados. Descrevendo esses aspectos matematicamente, buscou encontrar soluções ótimas para essas formulações. A solução para um detector ótimo de bordas encontrada por ele foi a utilização da primeira derivada de uma gaussiana, demonstrada na equação 2.5 (CANNY, 1986).

$$\frac{\mathrm{d}}{\mathrm{d}x} e^{-\frac{x^2}{2\sigma^2}} = \frac{-x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}} \tag{2.5}$$

A abordagem de Canny é comumente usada como um pré-processamento para a próxima a técnica de segmentação baseada na detecção de descontinuidade que será apresentada, a Transformada de Hough (TH). Essa transformada se enquadra nos algoritmos de detecção de pontos e linhas. Entretanto, ela pode ser utilizada para determinar qualquer forma geométrica facilmente parametrizadas como circunferências e parábolas.

2.1.5.3 Transformada de Hough

A Transformada de Hough (TH) é uma técnica de reconhecimento de formas geométricas facilmente parametrizadas como linhas, círculos e elipses. Pontos colineares ou pontos de uma determinada forma geométrica no plano da imagem podem ser mapeados em construções geométricas que se intercedem no espaço da transformada ou espaço de parâmetros (HART, 2009).

Foi inicialmente proposta por Hough (1962) como um método para reconhecer padrões em fotografias, em especial, para detectar o movimento de partículas subatômicas em fotografias de câmara de bolhas. Ela mapeava pontos colineares em retas que se intercedem em um único ponto no espaço de parâmetros.

Em 1969, Rosenfeld (1969) em seu livro *Picture Processing by Computer* definiu a transformada algebricamente, dada pela equação 2.6.

$$m = y_i - m * x_i \tag{2.6}$$

onde x_i e y_i são pontos da imagem e b e m são os eixos do plano da transformada, m é a declividade e b, a interseção com o eixo y.

A ideia de Hough e Rosenfeld, utilizando-se os parâmetros de declividade e interseção para identificar retas em imagens foi a base para Duda e Hart (1971) desenvolverem o modelo atual da Transformada de Hough. Esse modelo tornou-se popular após os avanços feitos por Dana H. Ballard (1981) na publicação Generalizing the Hough Transform to detect arbitrary shapes

A TH de Duda e Hart (1971) buscou solucionar o problema da proposta de Hough-Rosenfeld, cuja transformada é ilimitada para ambos os parâmetros. Isso amplia sua complexidade, isto é, nessa técnica, os pontos das retas aproximadamente paralelas ao eixo x, transformavam-se em retas cuja interseção ocorrem no infinito. A solução encontrada foi utilizar a equação normal da reta, equação 2.7, para parametrizar os pontos do plano da imagem. Dessa forma, uma reta é especificada pelo ângulo da normal e a distância algébrica a origem.

$$\rho = x_i cos(\theta) + y_i sen(\theta) \tag{2.7}$$

Esses parâmetros podem ser limitados de modo a garantir que cada reta no espaço da imagem possua um único ponto (ρ, θ) que a represente no espaço de parâmetros. Assim, essa transformada mapeia pontos colineares em curvas senoidais que se interceptam em um único ponto no espaço de parâmetros (HART, 2009). Em resumo:

- um ponto no plano da imagem corresponde a uma curva senoidal no plano dos parâmetros;
- um ponto no plano dos parâmetros corresponde a uma reta no plano da imagem;
- pontos sobre uma mesma reta no plano da imagem correspondem a curvas sobre um ponto comum no plano dos parâmetros;
- pontos sobre uma mesma curva no plano dos parâmetros correspondem a diferentes linhas sobre um único ponto no plano da imagem.

Duda e Hart (1971) ainda propuseram o uso do método para uso na detecção de outros padrões geométricos a partir de curvas de parametrização analíticas, como a detecção de círculos. Entretanto nesse projeto utilizou-se apenas a TH como detector de linhas.

2.1.5.4 Watershed

A segmentação tem por objetivo dividir a imagem em regiões. Os métodos apresentados anteriormente, buscaram encontrar fronteiras entre as regiões, como no caso dos detectores de borda e a TH, ou definir limiares baseando-se na distribuição de propriedades dos pixels, como a intensidade e as cores. Nessa seção será explicada as técnicas que buscam encontrar a região diretamente.

A segmentação por regiões pode ser divida em duas abordagens: segmentação por crescimento de região e a segmentação por difusão e fusão de regiões. A primeira agrupa os pixels ou sub-regiões em regiões maiores com base em critérios pré-definidos. Se inicia o

processo a partir de um conjunto de pontos "sementes"de onde a região começa a crescer anexando a cada semente pixels vizinhos que têm propriedades predefinidas semelhantes às das sementes. A seleção do critério de similaridade é variada e depende do problema em questão e do tipo de dados da imagem. A segunda abordagem busca em um primeiro momento subdividir a imagem em um conjunto de regiões distintas e arbitrárias, em seguida, fundir e/ou dividir as regiões de modo a satisfazer os critérios de segmentação (GONZALEZ; WOODS, 2006).

Conforme o autor supracitado, a técnica de Watershed (bacias hidrográficas) se aproveita de conceitos de todas as técnicas apresentadas. O seu conceito envolve a visualização da imagem em três dimensões, as coordenadas padrões e um terceiro eixo que representa a intensidade dos pixels. Nesse tipo de representação topográfica existem três pontos, os pontos de mínimo global, os pontos nos quais se uma gota de água caísse sobe eles, a gota iria deslizar e cair em dos pontos de mínima e os pontos de máxima onde se uma gota caísse ela teria igual probabilidade de escorregar para qualquer um dos pontos de mínima. Os dois primeiros pontos simbolizam bacias hidrográficas e serão considerados regiões de semelhança e o terceiro ponto formam as linhas de divisão ou linhas de watershed.

O objetivo do algoritmo de Watershed é encontrar as linhas de divisão. Para compreensão desse algoritmo normalmente se faz uma analogia em relação a imersão da topologia da imagem em um líquido. Considera-se que os pontos de mínimo são furos na topologia. Ao mergulhar a topologia o líquido entra pelo furo de mínimo e vai preenchendo as bacias, no momento que a água de duas bacias estão prestes a se juntar constrói-se uma barragem, correspondente as linhas de watershed. A inundação continua até chegar a um ponto em que só existirá o topo das barragens, que é o resultado da segmentação (PEDRINI; SCHWARTZ, 2008).

Uma abordagem para controlar a criação de barragens é baseada no conceito de marcadores. Um marcador é um componente conectado que pertence a imagem. Têm-se os marcadores internos associados aos pontos de interesse e os indicadores externos associados ao fundo.

2.1.6 Operadores Morfológicos

A morfologia matemática iniciou-se em 1964 com os pesquisadores franceses Georges Matheron e Jean Serr. Desde então, constitui de importante ferramenta no PDI, ganhando força por sua simplicidade de implementação (FACON, 2011).

A base dessa matemática está formulada sobre o elemento estruturante e sobre as operações dilatação e erosão. O elemento estruturante é a matriz que interage com os objetos da imagem alterando sua aparência e sua forma. As características mais importantes

desse elemento são a sua forma e seu tamanho. A morfologia matemática em sua busca avaliar se o elemento estruturante está ou não está contido no objeto da imagem.

As operações morfológicas mais básicas são: a erosão e a dilatação. A erosão remove *pixels* dos limites de um objeto na imagem, enquanto a dilatação adiciona *pixels* a essas bordas. Quando usadas em conjunto com as mesmas definições de vizinhança, *template* ou máscara, tem-se as operações abertura, erosão seguida de dilatação, e fechamento, dilatação seguida de erosão. A abertura é utilizada para remover pequenos objetos de uma imagem preservando a forma e o tamanho dos objetos grandes. Por sua vez o fechamento é utilizado para unificar elementos, eliminando os vazios entre eles (OPENCV, 2018).

Além da extração de ruídos e objetos indesejados, como também unificação e divisão de regiões, a morfologia matemática pode ser utilizada para extrair bordas de uma imagem. Para isso, faz uso do operador gradiente morfológico. Esse operador aplica sobre a imagem uma dilatação e isoladamente uma erosão, em seguida, efetua a subtração do resultado da dilatação pelo resultado da erosão. Observa-se que um dos efeitos da dilatação é a expansão do objeto, principalmente, em um cenário binário, quando esse é expresso pela cor branca e o fundo é preto. Já a erosão contrai a imagem nessa situação. Assim ao se efetuar a subtração obtém-se a região de fronteira do objeto.

Além disso, pode-se realçar o contraste de imagens em tons de cinza aplicando os operadores top-hat e bottom-hat. A operação top-hat é definida como sendo a abertura subtraída da imagem original, enquanto o bottom-hat é o fechamento subtraído da imagem original. A primeira destaca objetos claros sobre fundos escuros, enquanto que a segunda destaca objetos escuros sobre fundos claros. O realce de contraste é feito ao se aplicar a diferença entre o top-hat e o bottom-hat somado a imagem original (GONZALEZ; WOODS, 2006; PEDRINI; SCHWARTZ, 2008).

2.1.7 Representação

Após a segmentação objetos podem ser representados em termos de suas características externas (bordas) ou internas (pixels que o compõem). Nessa seção serão apresentadas duas abordagens de representação de objeto que buscam facilitar sua descrição, isto é, extração de características. Ambas as abordagem são representações baseadas em bordas, a primeira é aproximação por polígono e a segunda é o rastreador ou seguidor fronteiras também conhecido como contornos.

Em muitas situações as bordas obtidas não coincidem com o formato do objeto. O principal objetivo da aproximação poligonal é capturar a essência da forma da borda com o menor número possível de segmentos poligonais. Para isso existem diversas técnicas que podem ser computacionalmente bem custosas (PEDRINI; SCHWARTZ, 2008).

Os algoritmos rastreadores de fronteiras ordenam os pixels da bordas de uma região

em sentido horário ou antem horário. Esse procedimento é necessário para os algortimos extratores de características. Um algoritmo básico para essa técnica é o algoritmo de rastreamento de fronteira de Moore (1968). Além desse, outros algortimos foram propostos, por exemplo o de Suzuki et al. (1985) que é utilizado em uma das principais funções da principal biblioteca de visão computacional atual, o OpenCV.

2.2 Redes Neurais Artificiais

Os seres humanos não só são capazes de realizar decisões complexas, como também possuem a habilidade de aprender. Essa habilidade permite ao homem realizar tarefas cada vez mais complexas e encontrar formas distintas de alcançar seu objetivo. Quase sempre o processo de aprendizagem passa por um ciclo de treinamento, em que para se chegar a solução final são necessários muitas tentativas, certas e erradas, até se encontrar aquela que melhor se adapta ao problema. Esses processos só são possíveis graças ao processamento paralelo e distribuído da rede de bilhões de neurônios presente no cérebro (HAYKIN, 2007).

O funcionamento do neurônio pode ser estudado a partir de três funções específicas e complementares que dividem essa célula nas regiões: dendritos, corpo e axônio. Os dendritos são as estruturas capazes de receber os impulsos nervosos ou estímulos e enviá-las para o corpo, que processa essas informações. Ao processar um determinado número de impulsos, o corpo envia um novo impulso que será transmitido pelo axônio até a sinapse nervosa, onde dendritos de outros neurônios poderão captar esse estímulo. Esse processo pode se repetir por várias camadas de neurônios, dando ao cérebro a capacidade de receber entradas sensoriais e atribuir reações físicas para elas (FERNEDA, 2006).

Assim, o cérebro pode ser interpretado como um computador de alta complexidade, não linear e de processamento paralelo, com capacidade de organizar seus constituintes funcionais, neurônios, de forma a realizar diversos processos, como identificação, reconhecimento de padrões e controle de funções, e atuar de forma mais rápida e eficiente que os mais avançados computadores existentes (HAYKIN, 2007). Tal comparação levou a tentativas de construção de um modelo computacional que simula o cérebro humano. Essa busca se inicia na década de 40 com o trabalho de McCulloch e Pitts (1943), e em 1958 Rosenblatt propôs o método de aprendizagem para redes neurais artificiais Perceptron. Entretanto, nos finais dos anos 60, as dificuldades metodológicas e tecnológicas desse método tornaram-se evidentes e em 1969 Minsky e Pappert publicaram um livro apresentando as limitações do perceptron. Assim, na década de 70 o entusiasmo por pesquisas nas redes neurais diminuiu retornando apenas durante os anos 80, graças aos avanços metodológicos e tecnológicos (FERNEDA, 2006).

Hoje as Redes Neurais Artificiais (RNA) figura-se como um dos principais métodos

dos sistemas inteligentes. A RNA apresenta vantagens de robustez, generalização ou adaptabilidade e paralelismo. Ela é usualmente utilizada para o aprendizados e reconhecimento de padrões, processamento de sinais e imagens e controle adaptativo e preditivo.

Em razão de sua complexidade, as definições de Redes Neurais Artificiais são diversas. Em Haykin (2007) foi definida como um sistema massivamente paralelo e distribuído, composto por unidades de processamento simples que possuem uma capacidade natural de armazenar e utilizar conhecimento. São um conjunto de técnicas computacionais que apresentam um modelo semelhante a estrutura neural, capaz de simular seu comportamento e seus elementos básicos são as conexões, os neurônios, os padrões e suas funções.

2.2.1 Neurônio Artificial

Como no sistema nervoso, o processamento mais básico de informações ocorre em unidades simples chamadas de neurônio artificial ou nós. Um neurônio artificial é composto por três partes: seus pesos associados, representando as sinapses do sistema nervoso, a junção somadora e a função de ativação.

A junção somadora é uma combinação linear das entradas pelos pesos associados somado limiar b_k , esse limiar aumenta ou diminui a influência do valor da entrada do neurônio para a ativação do mesmo. A função de ativação tem como objetivo limitar a saída e introduzir a não-linearidade no modelo, possuindo quatro formas básicas:

- Degrau: define a saída do neurônio como sendo 0 ou 1;
- Linear: não altera a saída do neurônio;
- Sigmóide ou Logística: mantém a saída do neurônio sempre positiva no intervalo de 0 a 1;
- Tangente Hiperbólica: define a saída do neurônio entre -1 e 1.

Além dessas, existem diversas variações das funções de ativação. A função ReLu, por exemplo, é uma variação da função linear, sendo uma função rampa iniciada em 0 e no intervalo positivo é a função identidade, limitando a saída do neurônio ao intervalo de 0 a infinito. Outra função bastante utilizada é a Softmax que consiste de uma função de ativação que retorna o maior valor dentre múltiplas saídas, muito utilizada em redes classificadoras como uma variação da função de ativação sigmóide, uma vez que essa função, cujo intervalo de saída está entre 0 e 1, retorna a probabilidade de um objeto pertencer a uma determinada classe.

Os neurônios artificiais fazem o processo das entradas recebida pelas suas conexões localmente recebendo apenas um valor de entrada e têm como saída apenas um valor,

sendo essa saída uma função das saídas da camada anterior e os pesos das conexões que ligam um neurônio ao outro. O sinal transmitido entre neurônios é associado a um peso que está presente na conexão e modula o sinal (CARDON; MULLER, 1994). Esses pesos são responsáveis pela memorização do padrão da RNA, são eles que são ajustados durante o processo de aprendizagem do sistema.

2.2.2 Arquitetura da Rede Neural Artificial

Uma rede neural é um conjunto de camadas de neurônios, uma camada constitui um conjunto de neurônios que recebem sinapses de um mesmo ponto (da entrada da rede ou de outra camada) e geram sinapses para um único ponto (para a saída da rede ou outra camada). A arquitetura de uma rede de forma mais simples consiste nas definições do número de camadas da rede, da quantidade de neurônios por camada e do tipo de neurônio número de entradas, pesos, bias e função de ativação da rede. A camada de entrada, é constituída por neurônios diretos, ou seja, que apenas fazem a conexão entre o estímulo e a rede (KOVÁCS, 2002) e a camada de saída recebe os estímulos da camada intermediária construindo o padrão da resposta e estimar o erro entre a saída da rede e a saída desejada. Em algumas arquiteturas estão presentes as camadas ocultas, onde ocorre a maior parte do processamento de dados, são as camadas extratoras de características, os pesos dessa camada são uma codificação dos padrões de dados da camada de entrada.

Existe dois tipos básicos de arquitetura de Redes Neurais Artificiais: Adaline e Perceptron. A arquitetura Adaline consiste de um único neurônio com função de ativação linear que sua saída é uma combinação linear das entradas moduladas pelos pesos das suas conexões somado com o bias.É possível combinar esse neurônio com outros formando uma Rede Madaline.

A estrutura da arquitetura Perceptron, pode possuir múltiplas camadas internas o que aumenta sua capacidade de processamento e permite que ela seja utiliza em processos mais complexos. A rede Perceptron permite a escolha da função de ativação que se quer utilizar, sendo mais genérica que a Adeline. O número de camadas escondidas presentes na RNA depende da aplicação, não sendo recomendado, em qualquer caso, um número elevado de camadas escondidas.

2.2.3 Aprendizado da Rede Neural

O elemento da rede neural onde está armazenado o conhecimento é a conexão. Toda conexão possui um peso associado, esse peso modula o sinal de saída dos neurônios, e são modificados a cada aprendizagem da RNA, o processo acaba quando o erro de saída for zero. Esse erro vem da diferença entre a saída desejada e a saída da rede e é retro-propagado para as camadas escondidas da rede.

As Redes Neurais Artificiais não funcionam sem os dados de saída associados aos dados de entrada de treinamento (CARDON; MULLER, 1994). A cada saída está associado um erro, esse erro é utilizado para atualizar os pesos das conexões, de forma que a rede neural atinja a saída desejada. Sendo o erro retro-propagado pelas camadas da rede, é importante a escolha correta do número de camadas e do número de neurônios, números muito pequenos ou muito altos podem causar a não convergência da rede.

Encontrar os parâmetros certos para a rede é uma das maiores dificuldades encontradas no treinamento, não existe uma fórmula ou regra que se aplique a todas as situações, sendo mais utilizado o método de tentativa e erro. O modelo não pode ser muito rígido nem muito flexível ao ponto de modelar os ruídos presentes nos dados, o ideal seria a rede responder de acordo com as características dos dados de entrada, porém não deve ser precisamente igual.

Apesar de um maior número de camadas escondidas resultar em uma melhora no processamento da rede, de forma que possa ser usada em problemas mais complexos, não é recomendado um número muito alto, pois pode resultar em saturações da retro propagação do erro e em redes muito complexas e computacionalmente custosas.

O número de neurônios por camada também é uma das dificuldades do treinamento. Um número alto de neurônios pode causar a memorização dos dados de treinamento (overfitting) pela rede em vez de extrair as características gerais, enquanto um número baixo de neurônios pode fazer a rede levar tempo em excesso tentando encontrar uma representação ótima.

Existem algumas sugestões para determinar corretamente o número de neurônios de uma rede, entre as mais utilizadas recomenda-se definir o número de neurônios em função da dimensão das camadas de entrada e saída da rede ou utilizar um número de sinapses dez vezes menor que o número de exemplos disponíveis para treinamento (BISHOP et al., 1995).

Existem diversas formas de realizar o treinamento da rede, isto é, a atualização dos pesos. A forma mais utilizada de retro propagar o erro e definir os novos pesos é o gradiente descente.

O treinamento por gradiente descendente é mais utilizado para dados não linearmente separáveis, a sua ideia chave é usar a descida do gradiente para procurar o melhor vetor de pesos, isto é, aquele que alcança o ponto de mínimo global do modelo. Primeiramente determina-se um vetor de pesos que minimiza o erro, começando com um valor aleatório de pesos e modificando-o aos poucos. A cada passo do treinamento o vetor de pesos é alterado na direção de produz a maior queda ao longo da superfície de erro, o processo só é interrompido quando encontra-se o erro mínimo global. A limitação desse treinamento seria o fato de que ao encontrar um mínimo, o treinamento é interrompido, não

sendo esse mínimo necessariamente o mínimo global, mas o primeiro mínimo encontrado pela rede.

O gradiente descente faz parte de um conjunto de funções denominadas otimizadores. O papel dessas funções é otimizar os pesos da rede para encontrar o ponto de mínimo global do erro da predição. Além do gradiente descendente pode-se citar gradiente descendente estocástico, o AdaGrad, o Adam e o RMSProp.

3 Ferramentas de Software e Base de Dados

Esse trabalho se propõem a explorar diferentes técnicas de PDI utilizando a linguagem de programação Python com o apoio de bibliotecas de Visão Computacional como o OpenCV. Para isso utilizou-se da aplicação web Jupyter para unificar código e relatório em um único documento. Essa seção se propõem a explanar as ferramentas de software e a base de dados utilizada.

3.1 Ferramentas de Softwarwe

As principais ferramentas de software utilizadas são o Jupyter Notebook, a linguagem de programação python, a biblioteca de visão computacional OpenCV e a biblioteca de aprendizado de máquinas TensorFlow. Tais ferramentas serão explanadas abaixo.

3.1.1 A linguagem de programação Python

Uma das linguagem de programação mais utilizadas em todo mundo, a linguagem Python se caracterizou por apresentar uma sintaxe simples porém com grande capacidade sendo de fácil entendimento e de baixa complexidade para programação (HALTERMAN, 2011). Foi desenvolvida por Guido Van Rossum em 1989 com objetivo de criar uma ferramenta de programação que agilizasse o processo de criação de softwares em comparação com as existentes na época (CHUN, 2006) , de tal modo que o inventor caracterizou o Python como uma linguagem de prototipagem (ROSSUM; BOER, 1991).

O python é uma linguagem interpretada de alto nível e que pode ser utilizada de duas formas: Interativa e Scripts. Na forma, interativa trabalha como uma linguagem de linhas de comando podendo mostrar o resultados do comando efetuado quando o usuário pede para executá-lo. Na forma de script o código python é escrito e salvo em um arquivo (normalmente com extensão .py) e num momento posterior se utiliza o interpretador para executar o arquivo (DOWNEY, 2012).

A criação de Van Rossum foi bem sucedida e é utilizada para o desenvolvimento de softwares em diversas companhias como Google, Yahoo, CERN e Nasa (HALTERMAN, 2011). Por sua facilidade, programadores em todo mundo começaram a desenvolver pacotes que ampliaram as capacidades da linguagem. Devido a pacotes como Numpy e Scipy que facilitam o uso de operações com vetores de uma ou mais dimensões, essa biblioteca tornou-se popular nos algoritmos de inteligência artificial e aprendizado de máquinas (RASCHKA, 2015).

Esse grupo de algoritmos buscam extrair padrões em dados, que quase sempre são operados como vetores. Devido a isso, a grande diversidade de pacotes e a popularidade entre os pesquisadores da áreas de IA como a visão computacional, essa linguagem foi escolhida para a elaboração deste trabalho. A Tabela 1 a seguir lista alguns dos principais pacotes presentes na linguagem python e sua aplicabilidade.

Aplicabilidade	**Pacotes**
Vetores	numpy
Imagens	opencv, scikit, numpy
Aúdio	librosa
Machine Learning e Deep Learning	pandas, scikit, tensorflow, pytorch, keras
Textos	nltk, numpy, scikit
Visualização de dados	matplotlib, seaborn, scikit
Arquivos	glob, glob2, sys
Aplicações Web	django
Sistemas	sys, os

Tabela 1 – Listagem de bibliotecas python e sua aplicação.

3.1.2 OpenCV

A plataforma OpenCV consiste de um conjunto de bibliotecas de código livre para o desenvolvimento de softwares de Visão Computacional e Aprendizado de Máquinas. Ela foi desenvolvida originalmente pela Intel com o objetivo de fornecer uma infraestrutura comum para as aplicações de Visão Computacional tanto para pesquisa quanto para uso comercial. O OpenCV consta com mais de 2500 algoritmos otimizados que podem ser utilizados para detecção e reconhecimento de faces, identificação de objetos, classificação de ações humanas em vídeos, trajetória de movimentação de câmeras e de objetos, extração de modelos 3D de objetos, trabalhos com visão estéreo, entre outras ações relacionadas ao PDI e a Visão Computacional (BRADSKI, 2000; CULJAK et al., 2012).

Inicialmente, foi desenvolvido em linguagem C++, mas hoje tem suporte para Python, Java, interfaces MATLAB e pode ser executado nos sistemas operacionais Windows, Linux, Android and Mac. É utilizado por diversas empresas como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota (BRADSKI, 2000).

O OpenCV possui implementações para as mais diversas técnicas do PDI como: Transformação de espaço de cores, normalização, Transformada de Hough, diversos filtros de suavização, detectores de borda, registro, entre outras (CULJAK et al., 2012). Pela diversidade de algoritmos de PDI já implementados no OpenCV e por sua relevância na comunidade científica essa biblioteca foi escolhida como pacote base de visão computacional deste projeto. De tal modo, que o estudo de caso feito prioriza a apresentação da técnica de PDI e não sua implementação em algoritmos.

Além de tudo disso, utilizou-se o módulo de Deep Learning do OpenCV para importar o modelo da rede de reconhecimento de objetos YOLOv3 (REDMON; FARHADI, 2018), uma rede que obteve bons resultados em termos de acurácia e tempo de inferência. Essa rede foi utilizada para detectar carros nas imagens do dataset de modo a facilitar o reconhecimento das placas.

3.1.3 TensorFlow e Keras

É uma biblioteca de código aberto para interpretação e execução de algoritmos aprendizado de máquinas para pesquisa e para o uso comercial. Foi desenvolvida por pesquisadores da Google no intuito de criar uma interface de aprendizado de máquina que executasse com pouca ou nenhuma alteração em diferentes ambientes, isto é, pudesse ser executada tanto em aplicações mobile quanto em grandes servidores de GPUs (ABADI et al., 2015).

O TensorFlow é um sistema flexível por poder ser utilizado por uma diversidade de algoritmos. Ele é comumente utilizado para treinamento e inferência de modelos de redes neurais profundas (Deep Learning). De tal modo que pode ser utilizado em aplicações de reconhecimento de fala, visão computacional, robótica, processamento de linguagem natural, extração de informações geográficas, entre outras áreas (ABADI et al., 2015). Neste trabalho utilizou-se a API de detecção de objetos do TensorFlow para reconhecer carros nas imagem testadas, similar a utilização do YOLOv3.

O Keras é um conjunto de rotinas e métodos programados em Python e que funcionam sobre bibliotecas, como TensorFlow. O Keras atua como uma camada mais alto nível sobre o TensorFlow facilitando a programação de algoritmos de RNA e Deep Learning (CHOLLET et al., 2015). Atualmente, já foi incorporado ao TensorFlow sendo portanto um módulo dessa biblioteca. Nesse projeto utilizou-se o Keras para programar uma pequena rede que pudesse realizar o OCR das placas detectadas.

3.1.4 Jupyter

O Jupyter é um projeto sem fins lucrativos de código aberto criado em 2014 fruto da extensão do projeto IPython e cujo objeto é criar um suporte em diversas linguagens de programação para o desenvolvimento iterativo da ciência dos dados e da computação científica (JUPYTER STEERING COUNCIL, 2018). Focado principalmente em linguagem de programação python, o projeto Ipython consiste de um interpretador interativo para python, que permite entre outras coisas atuar como Shell, isto é a interface de usuário com o sistema operacional através de linhas de comando. Uma das evoluções desse projeto foi o surgimento de uma nova interface denominada Notebook, executada como uma interface web no navegador, permitiu a integração entre a execução do código, textos,

expressões matemáticas, gráficos, vídeos e qualquer forma de visualização de informações que o navegador é capaz de suportar. Outra vantagem do Notebook é que ele permite compartilhar o código e as anotações facilmente em diferentes formatos, como pdf e HTML (PÉREZ; GRANGER, 2007).

Assim, o projeto Jupyter visa expandir essa integração alcançada na linguagem Python com o IPython para outras linguagem de programação e tem o apoio de grande empresas e instituições como Google, Microsoft, Netflix, Universidade de Berkeley, Anaconda, Bloomberg, entre outras (JUPYTER STEERING COUNCIL, 2018).

A possibilidade de unificação entre texto e código fez com que este trabalho optasse pela utilização do Jupyter Notebook como interface para a programação em Python devido a essa capacidade de integração entre desenvolvimento do código, documentação, execução e comunicação dos resultados em uma única plataforma (JUPYTER TEAM, 2015). Além disso, com a possibilidade de uso em conjunto da linguagem de marcação textual *Markdown* e de escrita de textos matemáticos e científicos LaTex essa ferramenta possibilita a elaboração de guias e relatórios com boa qualidade visual, com a vantagem de ser integrada ao código desenvolvido.

3.1.5 OpenALPR

O OpenALPR é uma biblioteca para Reconhecimento Automático de Placas Veiculares escrita em C++, mas com versões para C#, Java, Node.js e Python. Ela analisa imagens e streams de vídeo buscando identificar a placa, isto é, fornecer na saída uma representação textual dos caracteres presentes na placa. Essa biblioteca funciona em diferentes cenários e para diferentes países, inclusive o Brasil. Pode ser utilizada em associação com bancos de dados, em vídeos e imagens isolados, linha de comandos (OPENALPR TECHNOLOGY, INC, 2017).

Nesse projeto, foi utilizado como OCR do nosso caso de uso de detecção de placa, de modo a validar a possibilidade de um OCR nas placas detectadas pelo algoritmo desenvolvido. Além disso, serviu de baseline para os resultados do estudo de caso. O algoritmo de OCR do OpenALPR utiliza a engine de OCR Tesseract desenvolvida pela Google e permite que essa engine seja treinada para novos modelos de placa.

3.2 Base de dados

A base de dados utilizada neste trabalho foi a SSIG License Plate Character Segmentation Database (SSIG SegPlate Database) construída pelo Smart Surveillance Interest Group (SSIG) da Universidade Federal de Minas Gerais em 2015. Ao todo são 2000 imagens de placas, num total de 101 veículos rastreados. As imagens são da frente dos veículos e foram obtidas com uma câmera estática (GONÇALVES et al., 2016).

As imagens da base são em resolução FullHD (1920x1080) em formato PNG (Portable Network Graphics). O tamanho médio dos arquivos é de 4.08MB totalizando 8.6GB. A base possui imagens de múltiplos frames por carro, ou seja, um único veículo possui mais de uma imagem. O motivo disso é a exploração de abordagens para utilização de informações redundantes e de escolha de frame ótimo para a segmentação. A média de frames por veículo é de 19,8, com desvio padrão de 4,14 (GONÇALVES et al., 2016).

O objetivo principal da base foi criar um banco de imagens com placas brasileiras e focado na tarefa de segmentação dos caracteres da placa. As anotações dessa base contêm a posição da placa, o seu valor textual e a posição de cada caractere.

Além dessa base, utilizou-se a base UFPR-ALPR Dataset da Universidade Federal do Paraná para validação dos algoritmos desenvolvidos em uma outra base. Essa base é composta de 1500 rastreios de veículos totalizando 1200 carros e 300 motos. Foram retiradas com o uso de GoPro Hero4 Silver, Huawei P9 Lite e um iPhone 7 Plus (LAROCA et al., 2018).

4 Desenvolvimento de algoritmos de Visão Computacional para ALPR

Sistemas de reconhecimento de placas são importantes em diversas aplicações de segurança, o que inclui o monitoramento do trânsito em rodovias, em ruas, na identificação de possíveis irregularidades, controle de acesso em cidades e estacionamentos, radares de velocidade, entre outras aplicações. O reconhecimento automático desse tipo de placa tem sido amplamente estudado devido ao grande número de aplicações práticas, entretanto, ainda existe espaço para o desenvolvimento de soluções que consigam lidar com os mais diversos cenários e situações, além disso, com o avanço nas técnicas atuais de visão computacional, surgem novos métodos para alcançar esse objetivo (KHAN et al., 2018; CORNETO et al., 2017).

A maioria das soluções atuais não são suficientemente robustas situações do mundo real, como exemplo, situações de cenários externos com grande variação de iluminação e condições climáticas (LAROCA et al., 2018). Nos cenários reais, enfrentam-se diversas naturezas de problemas como variações nas placas e fontes de ruídos. As placas não seguem nenhum padrão internacional, assim jurisdições diferentes podem alterar a fonte utilizada, a largura dos caracteres, o espaçamento entre eles, a cor da placa, entre outras características. No Brasil, passa-se por uma fase de troca de modelo de placas, visto que, atualmente o país está adotando o modelo de placas do Mercosul. Situações de ambiente externo, com carros em movimento são afetadas por diversas fontes de ruído, por exemplo sombras, reflexos em poças de chuva, variações luminosas durante o dia, diferentes reflectâncias nas placas, pouco contraste em placas antigas, baixa resolução de imagens, borramento devido a velocidade do veículo, entre outras situações (BULAN et al., 2017).

De modo geral, a maioria das abordagens atuais de detecção de placas veiculares para esse cenário, começam pela detecção do veículo na pista, após localizam a placa, segmentam seus caracteres e finalmente para reconhecê-los aplicam a técnica de reconhecimento ótico de caracteres (OCR - Optical Character Recognition) (GONÇALVES, 2016). Neste trabalho, foram criados Jupyter Notebooks que abordam diferentes técnicas para cada uma das etapas listadas acima, com exceção da detecção de veículos e dando maior abrangência para a tarefa de localização das placas. No caso da detecção de veículos, será apresentado posteriormente um tópico para utilização de redes neurais de detecção pré-treinadas e os resultados obtidos comparando os métodos com e sem o reconhecimento do veículo.

A metodologia adotada priorizou o uso de técnicas do Processamento Digital de Imagens ao invés de partir para abordagens de Aprendizado de Máquinas. Importante destacar que para o trabalho exposto, a obtenção de bons resultados não são sua prioridade,

visto que ele objetiva a apresentação da técnica de PDI usada em linguagem Python para servir como guia para uma futura disciplina de PDI no curso de Engenharia Elétrica da UFPB ou projetos na area.

Ressalta-se que nem todas as abordagens propostas obtiveram resultados significativos. Isso ocorreu devido a pouca robustez dos algoritmos devido a dificuldade do banco de dados utilizados, que apresenta variações de tamanho,cor,luminosidade e ângulos para as placas. Durante as abordagens, alguns objetos do fundo foram detectados juntamente com as placas, a fim de isolar foram propostas algumas soluções que serão apresentadas separadamente após a descrição da abordagens de localização.

Além desses métodos existem na literatura outras propostas de algoritmos que envolvem diferentes técnicas de PDI e Machine Learning, a citar o uso de Classificadores Haar em cascata, Máquinas de vetores de suporte (SVM) e técnicas com janelas deslizantes.

4.1 Abordagens de Detecção e Localização de Placas

As abordagens a seguir são responsáveis por detectar a presença de uma placa na imagem e tentar localizar a posição dela na imagem. Para indicar a posição é utilizado uma caixa de marcação ou caixa delimitadora (*Boundind box*) que é um vetor dado por $[x, y, w, h]$ em que (x, y) são as coordenadas do ponto superior esquerdo da caixa e w e h correspondem ao comprimento e a altura da caixa, respectivamente, a partir do ponto (x, y) no sentido para direita e para baixo.

4.1.1 Detectores de Borda com limiarização global e representação por rastreamento de bordas

Esta abordagem utiliza técnicas de detecção de bordas para marcar as linhas que formam a placa, devido ao seu formato retangular e aos caracteres em seu interior. Uma vez feito a detecção das bordas utiliza-se a limiarização e operadores morfológicos para gerar uma imagem binarizada que apresente a região da placa em destaque. Com isso, pode-se aplicar uma etapa de segmentação que faz uso do rastreamento de bordas proposto por Suzuki e Abe (1985) para encontrar os contornos da imagem. Esses contornos passam então por uma etapa de aproximação de poligonal que aproxima cada contorno detectado a uma forma geométrica. Os contornos aproximados para um quadrilátero são demarcados com *bounding boxes* e filtrados com base em características como o *Aspect Ratio* (AR) da placa. Para placas brasileiras esse valor é próximo de 0,32, com exceção das placas de motocicletas. Esse algoritmo foi desenvolvido para uma única imagem, em seguida foi avaliado sobre 20 imagens e seus parâmetros foram ajustados de modo a obter o melhor resultado para as 20 imagens.

O algoritmo descrito a seguir segue algumas etapas dos sistemas de visão computacional. É com base nessas etapas que as seções estão divididas. A etapa de aquisição não será explicitada, pois foram utilizadas imagens do banco de dados do SSIG da UFMG cujas informações de aquisição foram informadas anteriormente. As etapas iniciais do código (importação de bibliotecas, leitura de imagens e conversão em escala de cores) também estão demonstradas na Figura 2. Por fim, nesse Notebook foram utilizadas as bibliotecas: OpenCV, NumPy e Matplotlib.

```
# Importação de bibliotecas OpenCV, MatplotLib e Numpy
import cv2 # biblioteca de visao computacional OpenCV
import numpy as np # biblioteca de manipulacao de vetores e ferramentas matematicas e cientificas Numpy
import matplotlib as mpl # biblioteca de manipulacao e exibicao de dados Matplotlib
from matplotlib import pyplot as plt # Pyplot: modulo do Matplotlib para exibir as imagens
# utilizacao do matplotlib em formatacao para o Jupyter
%matplotlib inline
```

```
# Carregando as imagens e convertendo para escala de cinza
path = '../Data/' # caminho para a pasta das imagens
img_color = [] # lista de imagens coloridas
img = [] # lista de imagens em tons de cinza
plt.figure(figsize=(15,8)) # definição do tamanho da janela de exibição das imagens
# A cada iteração uma imagem é carregada e convertida para tons de cinza, ao passo que é armazenada
# nas listas acima
for i in range(0,20): #A pasta data possui 20 imagens, i ira variar de 0 a 19
    # a identacao defini o escobo do laco FOR
    # a  funcao append() adiciona um elemento a lista
    # a funcao do OpenCV imread() realiza a leitura do arquivo de imagem
    # a funcao do OpenCV cvtColor() é usada para conversão de espaco de cores no caso RGB para GRAY
    img_color.append(cv2.imread(path + str(i+1) + '.png'))
    img.append(cv2.cvtColor(img_color[i],cv2.COLOR_RGB2GRAY))
    #Exibicao das imagens
    plt.subplot(4,5,i+1) #Cria subplots na imagem que possuira 4 linhas e 5 colunas com elementos de 1 a 20
    plt.imshow(img[i],cmap='gray') # Exibe a imagem com mapa de cores Gray
    plt.xticks([]), plt.yticks([]) # Remove a exibição dos valores das dimensões das imagens nos eixos x e y
```

Figura 2 – Etapas iniciais do algoritmo.

4.1.1.1 Etapa de pré-processamento

Essa etapa consiste das técnicas de PDI para preparar a imagem para etapa de segmentação. As técnicas utilizadas foram o filtro espacial de suavização, equalização de histograma e aplicação de detectores de bordas.

O filtro de suavização foi utilizado para reduzir ruídos na imagem, reduzindo as transições brutas de contraste e atenuando ruídos de alta frequência. Essa técnica, apesar de atenuar as bordas da placa com o fundo, é fundamental para homogeneização de outras regiões que seriam consideradas como placas para as demais funções do algoritmo. Foram

testados os filtros de suavização por média, mediana e bilateral e os resultados foram bem similares. Assim optou-se pelo filtro da média por questões de optimização, umas que ele é computacionalmente menos custoso em comparação aos demais. A utilização desse filtro se dá pela função *cv2.blur()* que recebe como entrada uma imagem e as dimensões da máscara para convolução. O código utilizado para aplicar esses filtros e plotar o histograma da imagem original e da imagem filtrada podem ser vistos na Figura 3, nota-se que uma única imagem pode ser lista a partir da indexação de um elemento das listas criadas anteriormente.

Figura 3 – Algoritmo de filtro de suavização e plotagem de histograma.

```
# Aplicação de filtro de Suavização e equalização de histograma
im = img[0]
mpl.rcParams['figure.figsize'] = (20.0, 10.0) #outra forma de definir as dimensões da janela do MatplotLib
# dessa forma esse valor fica fixo para todas as figuras geradas
plt.subplot(2,3,1)
plt.imshow(im,cmap='gray')
plt.title('Original') # adiciona um titulo ao subplot

# Histograma
nbins = 64 # numero de barras do grafico de histograma
h, bin_edges = np.histogram(im.ravel(), nbins,(0,255)) # a funcao ravel() transforma a imagem em um vetor 1D
# apos calculo do histograma pela funcao histogram, prepara-se para desenhar o histograma atraves do plt.bar
w=256./nbins
bin_centers = bin_edges[1:]-(w/2)
plt.subplot(2,3,4)
plt.bar(bin_centers, h, width=w)
plt.title('Histograma Original')

# Filtros espaciais de suavização
plt.subplot(2,3,2)
#im = cv2.medianBlur(im,5) #mediana
im = cv2.blur(im,(5,5)) #média
#im = cv2.bilateralFilter(im,5,20,20) #bilateral
plt.imshow(im,cmap='gray')
plt.title('Filtro de Media')

# Histograma
h, bin_edges = np.histogram(im.ravel(), nbins,(0,255))
bin_centers = bin_edges[1:]-(w/2)
plt.subplot(2,3,5)
plt.bar(bin_centers, h, width=w)
plt.title('Histograma apos filtragem')
```

Após a suavização realizou-se a equalização do histograma como uma etapa de generalização do algoritmo, tornando-o mais robusto a diferentes iluminações e sombreamentos. Isso ocorre, pois, a equalização de histograma busca uma distribuição mais homogênea dos valores dos *pixels* na imagem. Utilizou-se uma forma automática de equalização de histograma com uso da função *cv2.equalizeHist()* que recebe uma imagem e devolve outra que representa a imagem de entrada com o seu histograma equalizado. Essa técnica automática nem sempre resulta no efeito de homogeneização da luz, ficando a parte do programador avaliar a eficácia do método. Na Figura 4 observa-se o algoritmo utilizado para equalizar o histograma e plotar o histograma equalizado.

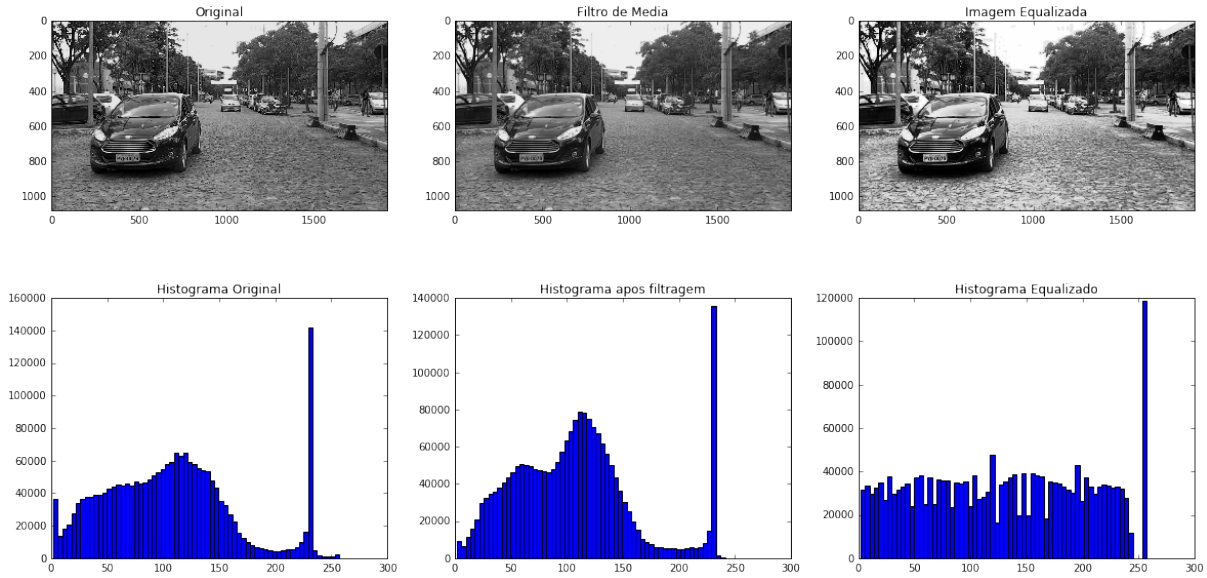
Nos algoritmos expostos preferiu-se utilizar para plotar o histograma a função do NumPy *np.histogram()*, a qual calcula o histograma de uma imagem, dividindo-a em faixas de valores (*bins*) e delimitando um intervalo de operação (*range*). Além disso, nos algoritmos, utiliza-se a função *np.ravel()* para transformar a imagem (vetor 2D) em um

Figura 4 – Algoritmo de equalização de histograma.

```
# Equalização
plt.subplot(2,3,3)
im = cv2.equalizeHist(im)
plt.imshow(im,cmap='gray')
plt.title('Imagem Equalizada')

# Histograma
h, bin_edges = np.histogram(im.ravel(), nbins,(0,255))
bin_centers = bin_edges[1:]-(w/2)
plt.subplot(2,3,6)
plt.bar(bin_centers, h, width=w)
plt.title('Histograma Equalizado')
```

vetor 1D. Os resultados da filtragem e da equalização de histograma podem ser observados na Figura 5.

Figura 5 – Etapas de pré-processamento para a abordagem 1 e avaliação com histogramas.



A detecção de bordas foi feita aplicando-se diferentes detectores de bordas. Ao final selecionou-se os filtros de Sobel conforme indicado pela literatura (ANAGNOSTOPOULOS, 2014). Os filtros aplicados nessa etapa foram Laplaciano, de Prewitt, de Sobel, a técnica de Canny e a operação morfológica Gradiente Morfológico. Os filtros de aguçamento de Prewitt e Sobel foram avaliados em ambos os eixos de forma individual e para o Sobel calculou-se ainda a magnitude e o ângulo. Ao final dessa etapa, trabalhou-se exclusivamente com os filtros de Sobel sobre o eixo x, eixo y e a magnitude, porém a abordagem completa fez uso apenas do Sobel no eixo x, por esse destacar as linhas verticais presentes na placa e nos seus caracteres, mas não tão presente no veículo. Na Figura 6 está representado a célula do *Jupyter Notebook* utilizada para esse teste.

Como as funções do OpenCV para aplicação dos filtros de Sobel e Laplaciano produzem na saída uma imagem representada por variáveis do tipo ponto flutuante (*floats*), para conseguir utilizar as demais funções do OpenCV recebendo elas como entrada é

Figura 6 – Algoritmo dos detectores de bordas.

```
# Filtro Laplaciano
laplacian = cv2.Laplacian(im,cv2.CV_64F)
laplacian = cv2.normalize(laplacian, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

# Filtro de Prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]]) # filtro de Prewitt eixo x
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]]) # filtro de Prewitt eixo y
img_prewittx = cv2.filter2D(im, -1, kernelx)
img_prewitty = cv2.filter2D(im, -1, kernely)

# Filtro de Sobel
sobelx = cv2.Sobel(im,cv2.CV_64F,1,0,ksize=5) # filtro de sobel eixo x
sobely = cv2.Sobel(im,cv2.CV_64F,0,1,ksize=5) # filtro de sobel eixo y
sobel_mag = (sobelx**2 + sobely**2)**0.5  # magnitude de Sobel
sobel_ang = np.rad2deg(np.angle(sobelx+sobely*1j)) # angulo das bordas de Sobel
sobel_ang = (sobel_ang >= 0)*sobel_ang + (sobel_ang < 0)*(sobel_ang + 180) # angulos entre 0 e 180
sobel_ang = (sobel_ang < 180)*sobel_ang # angulo de 180 = angulo 0

sobelx = cv2.normalize(sobelx, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
sobely = cv2.normalize(sobely, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
sobel_mag = cv2.normalize(sobel_mag, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
sobel_ang = cv2.normalize(sobel_ang, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

# Filtro de Canny
img_canny = cv2.Canny(im,100,200)

# Gradiente Morfológico
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
morphg = cv2.morphologyEx(im, cv2.MORPH_GRADIENT, kernel)
```

preciso normalizá-las de 0 a 255 e converter seu tipo para *uint8*. Para isso, utilizou-se a função do *cv2.normalize()*. Na Figura 7 visualiza-se os resultados obtidos após a aplicação de cada um dos detectores de bordas citados (as imagens originais foram recortadas para facilitar a visualização).

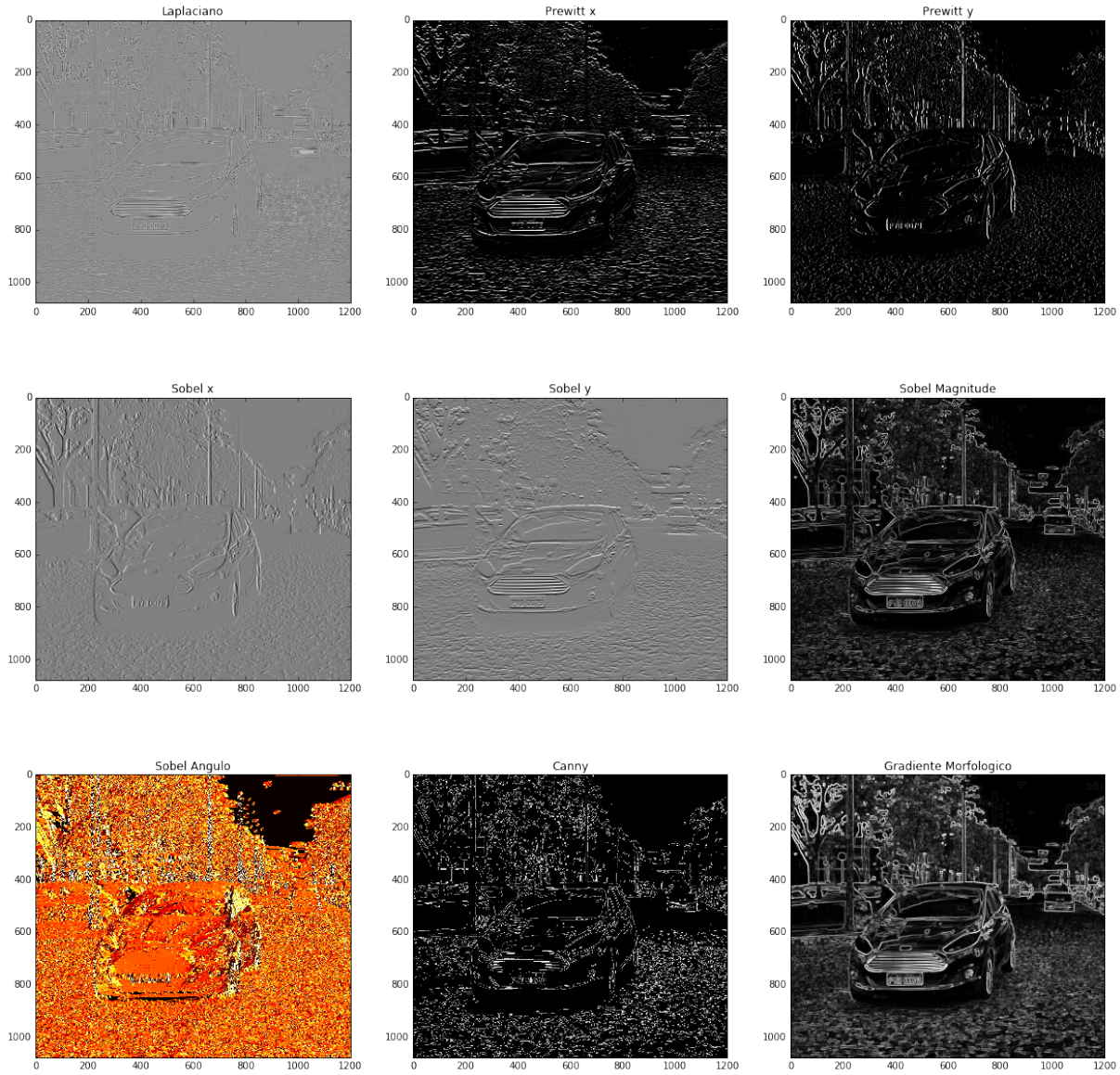
4.1.1.2 Etapa de segmentação

Nessa etapa toda a imagem foi segmentada com uso da técnica de limiarização em dois níveis. Essa etapa é necessária devido a aplicação do rastreador de bordas que se utiliza de imagens binarizadas. Para determinar o limiar testou-se três possibilidade: Teste empírico, Otsu e limiarização adaptativa com uso de distribuição gaussiana e média.

O método empírico foi aplicado utilizando-se um laço de *FOR*, variando o limiar e analisando a imagem qualitativamente. Essa análise consistiu em observar a imagem e buscar reduzir o número de elementos ruidosos destacados, porém com o menores efeitos sobre a região da placa. O método de Otsu foi implementado com a função *cv2.threshold()* e os métodos adaptativos com a função *cv2.adaptiveThreshold()*. Foi feita uma tentativa de seleção do limiar por meio da análise qualitativa do histograma, porém os histogramas apresentados não apresentaram o formato ideal para isso, ou seja, picos separados por vales. Na Figura 8 estão exibidos os algoritmos para os métodos de limiarização testados.

Após a comparação dos resultados obtidos aplicou-se um limiar global, cujo valor foi obtido pelo método empírico. O uso de uma valor fixo para o limiar reduz a robustez do algoritmo, uma vez que, ele não consegue se adaptar para novas imagens, isto é, não existe um cálculo automático para computar o valor ótimo para cada imagem. Os resultados para

Figura 7 – Aplicação de filtros de aguçamento.



essa limiarização aplicada nas imagens dos gradientes de Sobel x, Sobel y e Magnitude de Sobel podem ser vistos na 9.

Observou-se, porém, que a limiarização para reduzir o ruído (pontos brancos na imagem, que não estão na região da placa) através do destaque das bordas mais fortes, fez com que as linhas que demarcam as bordas da placa ficassem atenuadas. Para corrigir esse efeito, adicionou-se uma etapa de pós-processamento que faz uso de operadores morfológicos para destacar a região de interesse.

Com o objetivo de expandir as regiões brancas da placa, utilizou-se uma estrutura morfológica de formato retangular, devido à forma da placa, e testou-se os operadores dilatação, fechamento, tophat e gradiente morfológico. Os testes foram feitos com diferentes tamanhos de estrutura e combinações entre os métodos. Ao final observou-se que a dilatação

Figura 8 – Algoritmos de segmentação por limiarização.

```
for i in range(1,10):
    limiar = int(255*i/10.0)
    _,xthr = cv2.threshold(sobelx,limiar,255,cv2.THRESH_BINARY)
    _,ythr = cv2.threshold(sobely,limiar,255,cv2.THRESH_BINARY)
    _,mthr = cv2.threshold(sobel_mag,limiar,255,cv2.THRESH_BINARY)
```

(a) Teste empírico com limiarização global.

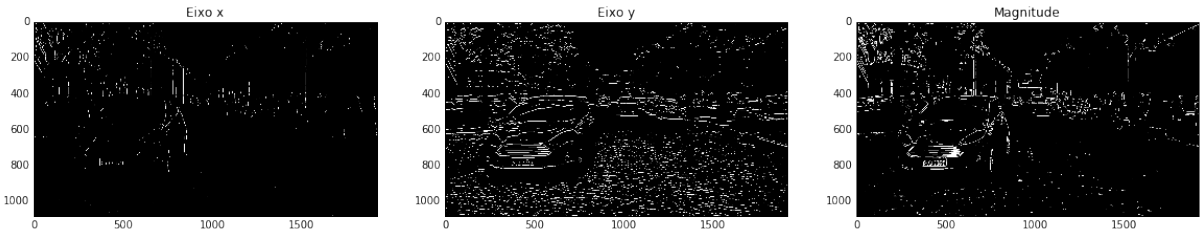
```
_,xthr = cv2.threshold(sobelx,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
_,ythr = cv2.threshold(sobely,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
_,mthr = cv2.threshold(sobel_mag,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

(b) Teste com limiarização por Otsu.

```
xthr = cv2.adaptiveThreshold(sobelx,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,3,0)
ythr = cv2.adaptiveThreshold(sobely,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,3,0)
mthr = cv2.adaptiveThreshold(sobel_mag,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,3,0)

plt.subplot(1,3,1), plt.imshow(xthr,cmap='gray'), plt.title("Eixo x"), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2), plt.imshow(ythr,cmap='gray'), plt.title("Eixo y"), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3), plt.imshow(mthr,cmap='gray'), plt.title("Magnitude"), plt.xticks([]), plt.yticks([])

plt.figure(figsize=(15,10))

xthr = cv2.adaptiveThreshold(sobelx,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,3,0)
ythr = cv2.adaptiveThreshold(sobely,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,3,0)
mthr = cv2.adaptiveThreshold(sobel_mag,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,3,0)
```

(c) Teste com limiarização adaptativa.

Figura 9 – Limiarização global sobre o gradiente de Sobel para os eixos x, y e para sua magnitude.

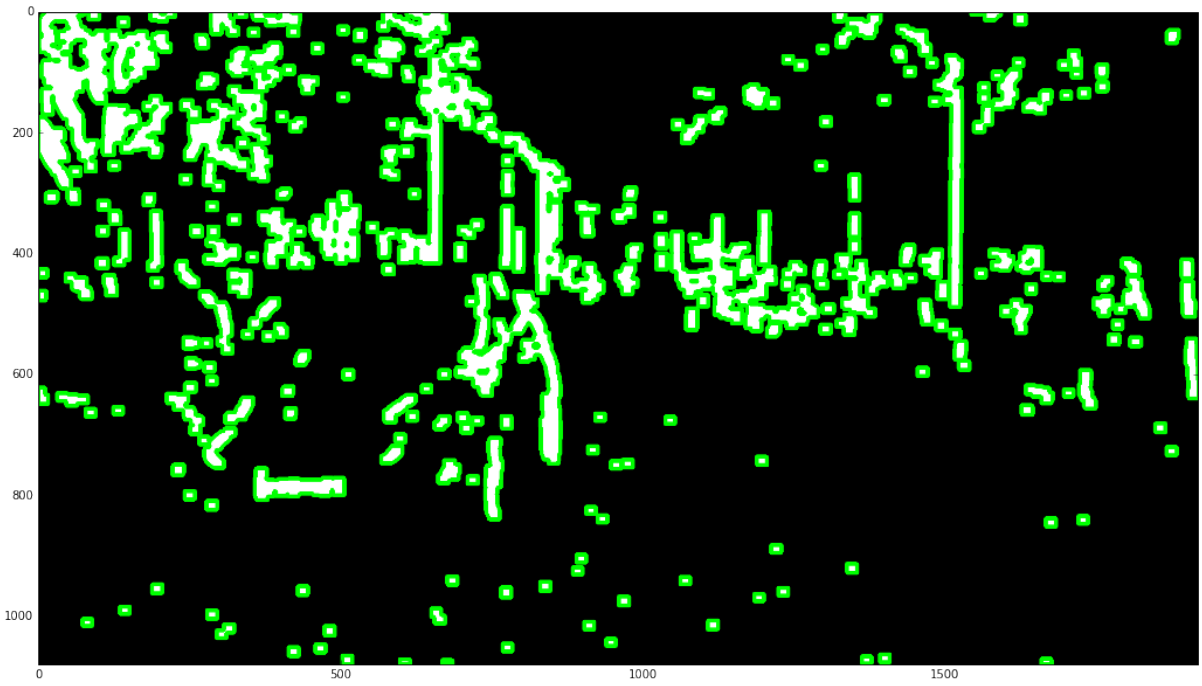


com estruturas de tamanhos próximos 17x13 obtiveram bons resultados em comparação aos demais. Observou-se que a melhor imagem para essa abordagem foi a imagem do gradiente de Sobel sobre o eixo x. O resultado dessa etapa é apresentado na Figura 10, enquanto que o algoritmo de para implementação da dilatação pode ser visto na Figura 11.

4.1.1.3 Etapa de extração de características

Nas etapas anteriores os parâmetros utilizados buscaram fortalecer características da placa como a variação de contrastes e seu formato retangular, entretanto é na etapa de extração de características que a busca pelas propriedades do objeto se intensificam com o objetivo de segmentá-lo completamente da imagem de fundo. Nessa etapa, utilizou-se uma abordagem de rastreamento de bordas, ou seja, busca de contornos com base na técnica proposta por Suzuki et al. (1985) e implementada no OpenCV na função *cv2.findContours*.

Figura 10 – Aplicação de filtro morfológico de dilatação sobre o gradiente de Sobel e aplicação de técnica de busca de contornos.



Na Figura 10 também está representado pelas linhas verdes o resultado da aplicação dessa técnica na imagem obtida após a a operação de dilatação da etapa anterior e na Figura 11 o algoritmo para aplicação dessa técnica de representação.

Figura 11 – Algoritmo da operação morfológica dilatação e da técnica de busca de contornos.

```
# Definição do elemento estruturante
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(17,13)) # MORPH_RECT devido ao formato da placa
# Aplicação da Operação Morfológica Dilatação
morphDx = cv2.dilate(xthr,kernel,1)
# Algoritmo de Busca de Contornos
_,contours1, hierarch = cv2.findContours(morphDx, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# Desenhando os contornos e plotando a imagem
img_cont1 = cv2.cvtColor(im,cv2.COLOR_GRAY2BGR) # nova imagem colorida a partir da de tons de cinza
cv2.drawContours(img_cont1, contours1, -1, (0,255,0), 5) # desenho de todos os contornos (indicado pelo -1)
plt.imshow(img_cont1,cmap='gray'),plt.xticks([]), plt.yticks([])
```

O próximo passo se utiliza do algoritmo de *Ramer–Douglas–Peucker* para aproximar os contornos a um polígono. No OpenCV esse algoritmo é implementado na função *cv2.approxPolyDP()*. Os contornos aproximado para um quadrilátero são selecionados e cria-se uma caixa de marcação (*Bounding Box*) em torno dele. Utilizando-se as dimensões dessa caixa calcula-se seu *Aspect Ratio* (AR) como sendo a razão entre a altura e o comprimento da caixa. Aqueles que apresentarem AR dentro de uma faixa de valores próximos a 0,32 (nessa abordagem, para as 20 imagens utilizou-se 0,3 a 0,45) foram classificados como possíveis candidatos a placas. Na Figura 12(a) visualiza-se as caixas de marcação obtidas após a aproximação dos contornos e seleção dos quadriláteros em

vermelho, e em verde os candidatos a placa após a seleção por meio do AR e na Figura 12(b) o algoritmo de implementação dessa etapa.

Figura 12 – Filtragem de contornos por aproximação de contornos e AR.

(a) Resultado final após a abordagem de detecção 1. Em verde as regiões eleitas como possíveis candidatos a placa.

```
color = cv2.cvtColor(im,cv2.COLOR_GRAY2BGR) # Imagem colorida gerada a partir de um em tons de cinza

#Iteração sobre casa contorno presente no vetor de contornos contours1
for cnt in contours1:
    #Algoritmo de aproximação de contornos do OpenCV
    epsilon = 0.05*cv2.arcLength(cnt,True) # valor de epsilon determinado empiricamente
    approx = cv2.approxPolyDP(cnt,epsilon,True) # função de aproximação de contornos
    #cv2.drawContours(color, approx,-1, (0,0,255), 10)
    if(len(approx) == 4): # selecionando apenas os contornos aproximados por quadrilateros
        x,y,w,h = cv2.boundingRect(cnt) #criação de caixas delimitadoras
        cv2.rectangle(color,(x,y),(x+w,y+h),(255,0,0),2) # desenha retangulos na imagem
        ar = 1.0*h/w # calculo do Aspect Ratio
        #cv2.putText(color,str(round(ar, 2)),(x,y),cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
        if(ar >= 0.3 and ar <= 0.4 and h*w >= 2300 and h*w <= 23000): # Filtragem por AR, área max e min
            cv2.rectangle(color,(x,y),(x+w,y+h),(0,255,0),8)
plt.imshow(color,cmap='gray'),plt.xticks([]), plt.yticks([])
```

(b) Algoritmo de aproximação de contornos e filtragem por AR.

Pela Figura 12 observa-se ainda que algumas regiões que não são placas foram selecionadas como possíveis candidatos a placa. Para essa imagem técnicas que restringem a área mínima da placa ou que definem uma posição na imagem onde as placas podem aparecer já seriam suficientes para eliminar os ruídos. Entretanto, em algumas ocasiões separar o candidatos ruins do candidato verdadeiro não é tão simples, e exigem técnicas mais complexas. Para esse problema foram propostas algumas soluções que serão discutidas posteriormente.

4.1.2 Matemática Morfológica com limiarização por Otsu e representação por rastreamento de bordas

Em resumo, essa abordagem utiliza apenas operadores morfológicos para auxiliar a função de rastreamento de bordas. É utilizado a operação *TOPHAT* após equalização de histograma como etapa de pré-processamento. Em seguida é segmentada com a seleção de limiar por Otsu e após sofre ação dos operadores abertura e fechamento. Por fim, aplica-se a função de detecção de contornos e filtra-se os candidatos de modo similar a etapa anterior. Esse algoritmo também foi desenvolvido para uma única imagem, em seguida foi avaliado sobre 20 imagens para ajustes de seus parâmetros e sua generalização. Na Figura 13 está representado o algoritmo utilizado para executar a segunda abordagem de localização de placas sobre as 20 imagens de ajustes de parâmetro.

Figura 13 – Algoritmo de localização de candidatos abordagem 2.

```
mpl.rcParams['figure.figsize'] = (20.0, 60.0) #dimensões da janela do MatplotLib
minArea = 2700 #definição da menor área de uma placa, baseado na menor placa de posível leitura
maxArea = 20000 #definição da maior área, baseado na maior placa encontrada (mais próxima da câmera)
#iteração sobe as 20 imagens
for i in range(0,20):
    im = img[i] # leitura de uma única imagem por vez
    im = cv2.equalizeHist(im) # equalização de histograma
    # Definição do Elemento Estruturante para o TOPHAT
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(17,17)) # MORPH_RECT devido ao formato da placa
    # Aplicação da operação morfológica TOPHAT
    morphTH = cv2.morphologyEx(im,cv2.MORPH_TOPHAT,kernel)
    # Limiarização por Otsu
    _,thr = cv2.threshold(morphTH,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    # Definição do Elemento Estruturante para a ABERTURA
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(1,11)) # MORPH_RECT devido ao formato da placa
    # Aplicação da operação morfológica ABERTURA
    morphAx = cv2.morphologyEx(thr,cv2.MORPH_OPEN,kernel)
    # Definição do Elemento Estruturante para o FECHAMENTO
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(23,1)) # MORPH_RECT devido ao formato da placa
    # Aplicação da operação morfológica Fechamento
    morphFx = cv2.morphologyEx(morphAx,cv2.MORPH_CLOSE,kernel)
    # Busca de contornos
    _,contours1, hierarch = cv2.findContours(morphFx, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    color = cv2.cvtColor(im,cv2.COLOR_GRAY2BGR) # imagem colorida a partir de uma em grayscale para exibição
    # iteração bre todos os contornos no vetor contours1
    for cnt in contours1:
        # Aproximação Poligonal
        epsilon = 0.05*cv2.arcLength(cnt,True)
        approx = cv2.approxPolyDP(cnt,epsilon,True)
        # Seleção de contornos aproximados por quadrilátero
        if(len(approx) == 4):
            # Criando as caixas delimitadoras e calculando o Aspect Ratio
            x,y,w,h = cv2.boundingRect(cnt)
            ar = 1.0*h/w
            #cv2.rectangle(color,(x,y),(x+w,y+h),(0,255,0),1)
            #cv2.putText(color,str(round(ar, 2)),(x,y),cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
            if(ar >= 0.3 and ar <= 0.55 and h*w > minArea and h*w < maxArea): # Seleção por características
                cv2.rectangle(color,(x,y),(x+w,y+h),(255,0,0),2)
    plt.subplot(10,2,i+1)
    plt.imshow(color,cmap='gray')
```

4.1.2.1 Etapa de pré-processamento

Diferentemente da abordagem anterior, nessa abordagem não é utilizado os filtros de suavização. O primeiro passo é a equalização automática do histograma que antecede o uso do operador morfológico *TOPHAT*. Essa operação é utilizada para destacar os elementos que são mais claros que os elementos ao seu redor, como é o caso das placas. Além disso, tem a característica de preservar os objetos cujas dimensões são inferiores ao

elemento estruturante utilizado. Nessa abordagem um elemento estruturante retangular de 17 *pixels* por 17 *pixels* é utilizado. A imagem resultante dessa etapa é exibida na Figura 14.

Figura 14 – Pré-processamento da segunda abordagem de detecção: Equalização de histograma e operador morfológico TOPHAT.

4.1.2.2 Etapa de segmentação

Essa abordagem aplicou a técnica de limiarização em dois níveis para realizar uma segmentação global na imagem, com o intuito de utilizar a técnica de rastreamento de bordas para imagens binárias. Nessa abordagem a seleção do limiar não se dar por meio de um limiar fixo, mas é calculada automaticamente para cada imagem por meio do método de Otsu que testa todos os valores limiar e escolhe aquele que melhor separa o fundo do objeto com base em dados estatísticos. O resultado para aplicação da limiarização por Otsu é ilustrado na Figura 15.

Em seguida é realizado uma etapa de pós-processamento que busca eliminar o excesso de pontos brancos no solo sem afetar a placa. Para isso utilizou-se dos operadores morfológicos abertura e fechamento. A abertura atua criando buracos nas regiões brancas onde existem pontos pretos próximas a elas. O fechamento por sua vez unifica regiões brancas eliminado as regiões pretas que as dividem.

Nessa abordagem se utiliza a abertura somente no eixo y da imagem definindo um elemento estruturante retangular de tamanho 1px por 11px e o fechamento somente no eixo x com elemento estruturante retangular de dimensões 23px por 1px. A escolha

Figura 15 – Limiarização por Otsu na segunda abordagem de detecção.

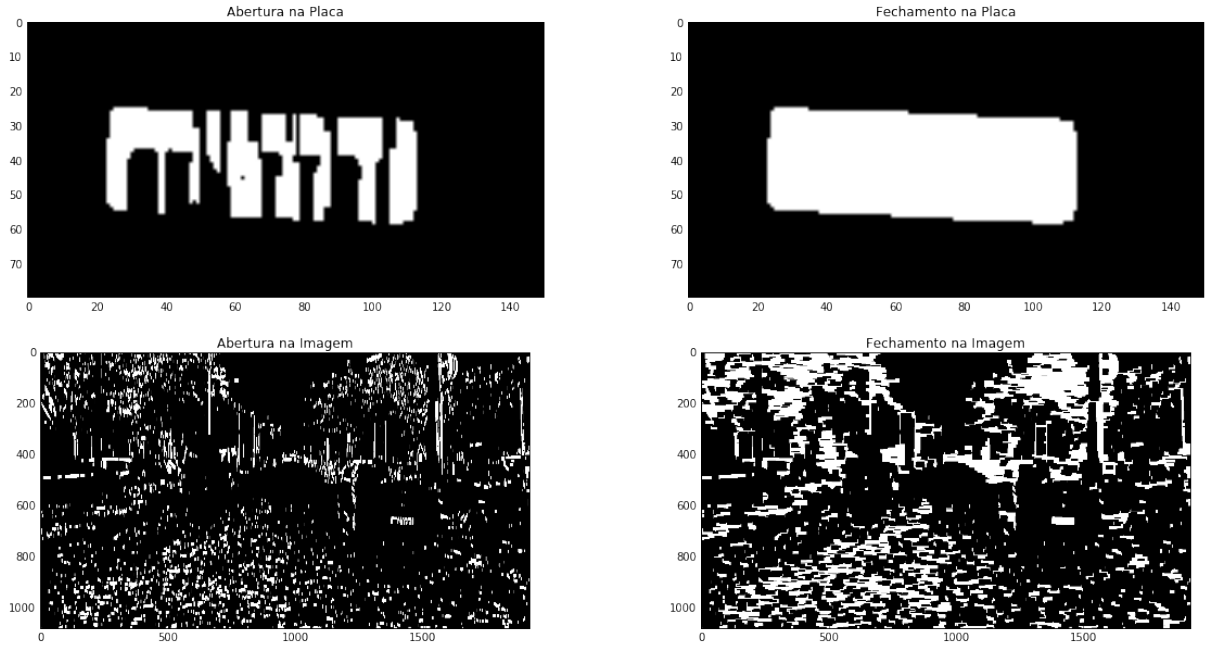
por trabalhar em apenas em um único eixo para cada uma das operações foi devido a observação que ao se realizar a abertura a imagem da placa era bastante afetada e o fechamento não conseguia atenuar esse efeito. Ao se fazer apenas nessas dimensões, durante a abertura os pontos escuros da placa onde estão os caracteres aumentam de tamanho, nesse caso somente no eixo y, dividindo a placa em partes brancas e pretas, em contrapartida os elementos do solo são eliminados. No fechamento sobre o eixo x as partes pretas que dividiam a placa são preenchidos pelo crescimento horizontal das partes brancas, em relação ao solo algumas das formas destruídas pela abertura são recuperadas porém em menor quantidade. O resultado desse processo no solo e na placa pode ser visto na Figura 16.

4.1.2.3 Etapa de extração de características

Essa etapa é similar a etapa de extração de características da abordagem anterior, seguindo o mesmo passo a passo. Entretanto ao observar a imagem de entrada vinda da etapa de segmentação observa-se que as imagens dessa segunda abordagem possuem mais pontos que não placa destacados. Isso reflete no resultado dessa etapa que passa a classificar um maior número de possíveis candidatos a placa. Na Figura 17 observa-se o resultado da busca por contornos.

Para atenuar esse efeito, como indicado na abordagem anterior, acrescentou-se na seleção dos candidatos características de área máxima e área mínima permitida. Essas áreas foram determinadas de forma manual observando o tamanho de uma placa muito

Figura 16 – Aplicações das operações Abertura e Fechamento e seus efeitos sobre a placa e sobre toda a imagem.



afastada da câmera mais que ainda fosse possível fazer sua leitura, para a área mínima e o tamanho de uma placa muito próximo da câmera para a área máxima.

Na Figura 18 observa-se a saída dessa etapa. Em vermelho os contornos aproximados a retângulos encontrados e em verde aqueles que foram selecionados como candidatos a placa. Para esse imagem o resultado não apresentou nenhum falso positivo, entretanto em algumas das 20 imagens, os algoritmos para seleção da verdadeira placa são necessários.

As duas próximas abordagens consistem em utilizar outras técnicas de segmentação. No caso para abordagem 4 usou-se a Transformada de Hough e na abordagem 5 a segmentação por regiões com a técnica de Watershed implementada pelo OpenCV. Nenhuma das duas abordagens obtiveram bons resultados quando testados nas 20 imagens, ou seja, não se mostram robustas a mudança das imagens.

4.1.3 Segmentação de linhas e extração de características por Transformada de Hough

Para essa abordagem foram testados as etapas de pré-processamento e segmentação das duas abordagens anteriores e substituindo na etapa de extração de características o rastreamento de bordas pela transformada de Hough (TH). Assim foram avaliados dois algoritmos: abordagem um com TH e abordagem dois com TH. Em todas as abordagem restringiu-se a encontrar linhas horizontais e verticais, desconsiderando as linhas oblíquas.

Entretanto antes de utilizar as abordagens anteriores, fez-se um teste com o método

Figura 17 – Rastreamento de contornos na segunda abordagem.

sugerido pelo OpenCV para trabalhar com Transformada de Hough que utiliza o detector de bordas Canny para gerar a imagem de gradiente a ser utilizada pela TH. De modo similar, acrescentou-se um detector de bordas ao final da etapa de segmentação de ambas as abordagens para gerar uma imagem de gradiente optimizando a TH. O detector de bordas utilizados para essa abordagem foi a operação morfológica denominada Gradiente Morfológico.

A função de TH utilizada foi a transformada de Hough probabilística (MATAS; GALAMBOS; KITTLER, 2000) implementada em *cv2.HoughLinesP()*. Essa TH permite limitar o tamanho das linhas encontradas, restringido-a a placa. O resultado para cada um dos métodos citados podem ser vistos na Figura 19, a letra (a) dessa imagem é a aplicação da Transformada de Hough sobre o operador de aguçamento. Na imagem (b) a aplicação de TH após a abordagem1. Por fim, na (c) é a aplicação dessa transformada após a segunda abordagem.

4.1.4 Segmentação por regiões com uso do método de Watershed

A técnica de segmentação por região implementada nesta abordagem consiste em unificar regiões similares. A técnica de Watershed como visto no capítulo é amplamente utilizada na visão computacional e possui diversas formas de implementação. A forma utilizada aqui será a adotada pela função do OpenCV *cv2.watershed()*.

Essa função utiliza necessita de marcadores que servirão como sementes para iniciar

Figura 18 – Resultado da segunda abordagem. Em verde as regiões classificadas como candidatos a placa.

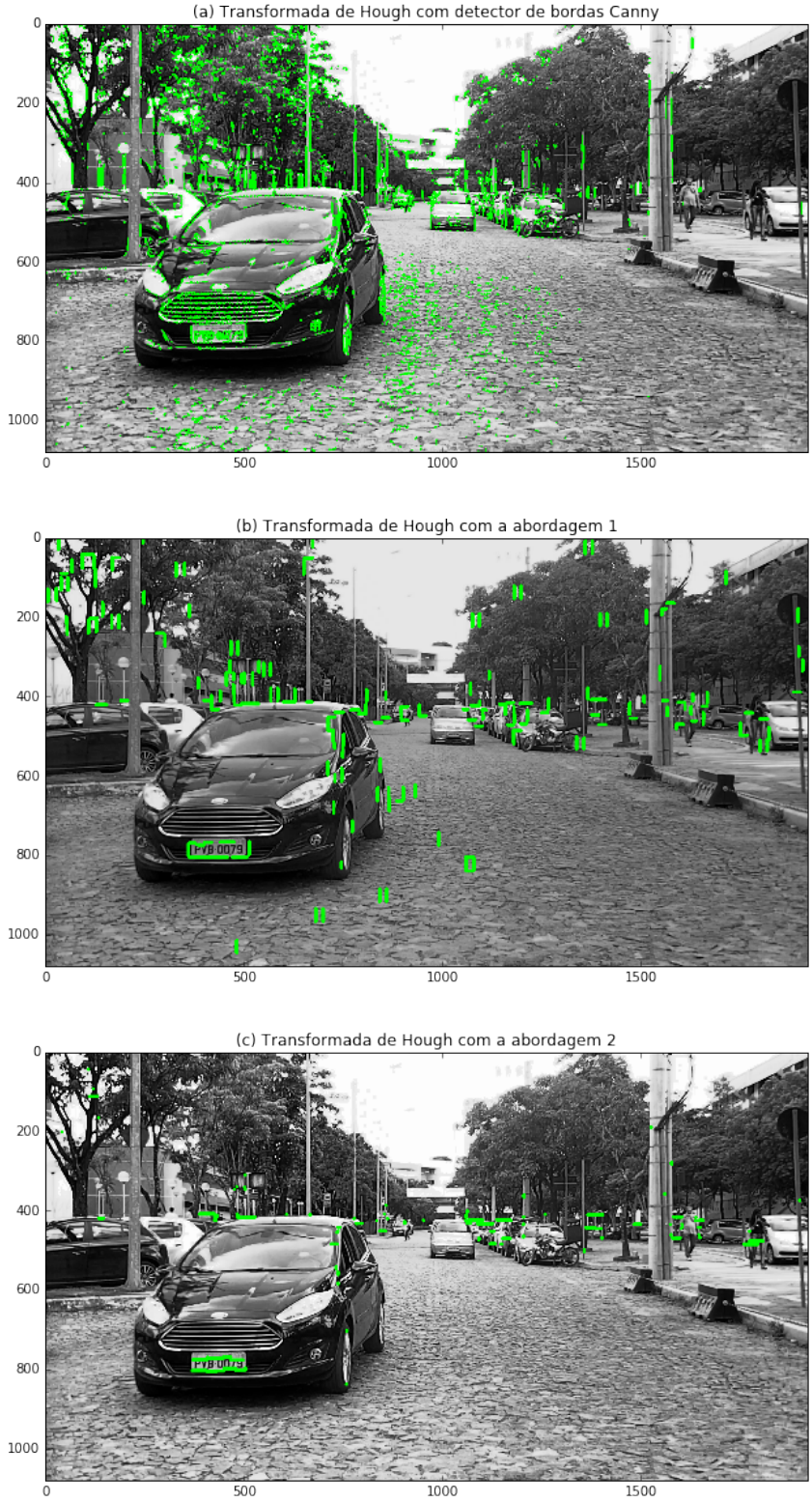
a busca por regiões de similaridade. Esses marcadores, porém, precisam ser definidos automaticamente e para isso utilizou-se uma abordagem que faz uso dos operadores de abertura e fechamento para separar o objeto de interesse do fundo e eliminar falso pontos de mínimos e máximos locais. A imagem obtida após essa etapa pode ser visualizada na Figura 20. Esse processo é similar ao efetuado na abordagem 2, inclusive utilizando o operador Tophat e a limiarização por Otsu.

Após essa etapa aplica-se a transformada distância que cria uma escala na região fazendo com que ela saia do seu ponto mínimo nas bordas ao seu ponto máximo no interior de forma gradual facilitando a detecção dos pontos de máxima e eliminando pequenos pontos de ruídos. A imagem então sobre uma limiarização utilizando Otsu e em seguida passa para a etapa de seleção dos marcadores do fundo da imagem.

É realizada uma subtração entre a imagem binarizada com a imagem daquilo que se quer encontrar, dado pela transformada distância. Com isso cria-se uma vetor de marcadores cujos elementos de interesses, isto é, as sementes, são marcadas com o valor 1 e os marcadores do fundo, resultado da subtração, são marcados com zero. Para melhorar os resultados obtidos, utilizou um recorte na imagem de forma a considerar as regiões passíveis de ocorrência de placa.

Na Figura 21 observa-se as duas imagens resultantes dessa seleção de marcadores. À esquerda os marcadores dos elementos de interesse resultante da Transforma distância e à direita o resultado da subtração entre a imagem após a primeira limiarização e os

Figura 19 – Transformada de Hough após a aplicação do detector de bordas Canny, da abordagem de detecção número um e da abordagem de detecção número dois, respectivamente.



objetos de interesse, indicando as fronteiras com o fundo.

O vetor de marcadores passa então pela função de *cv2.connectedComponent()* que

Figura 20 – Etapa de pré-processamento da quarta abordagem: equalização de histograma, tophat, limiarização por Otsu, abertura e fechamento.

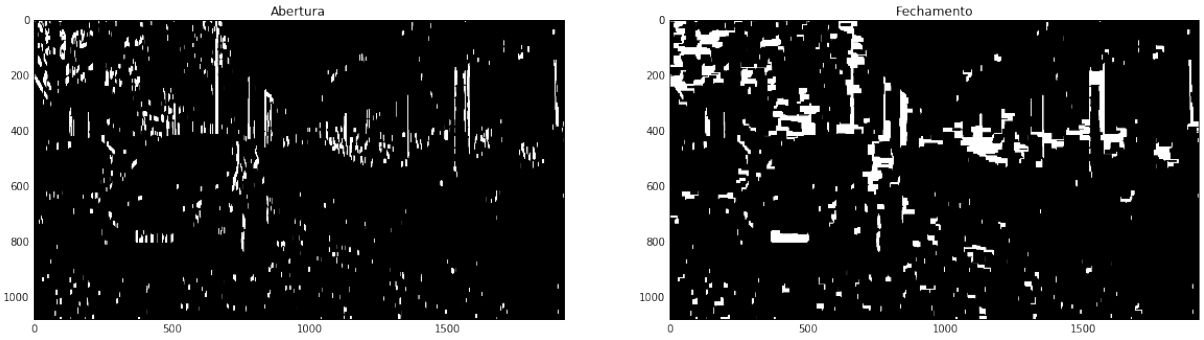
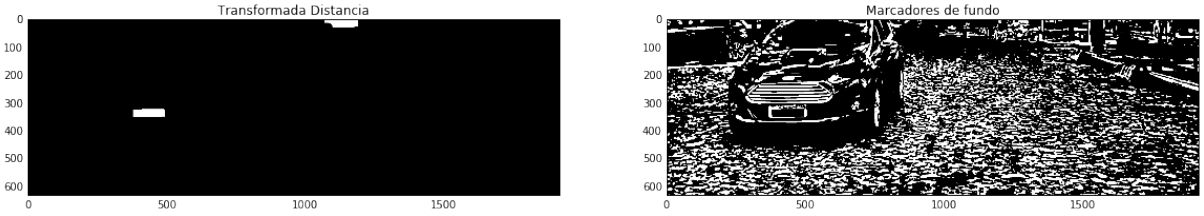


Figura 21 – Imagens para criação de marcadores. A da esquerda são os pontos de interesse e a da direita as fronteiras com o fundo.



rotula os elementos conexos distinguindo cada conjunto conexo. A matriz de marcadores rotulada é aplicada como parâmetro da função de Watershed e o resultado dessa segmentação pode ser visualizado na Figura 22. Nessa imagem tem-se em vermelho o resultado real do Watershed e em verde uma caixa delimitadora para facilitar a visualização.

Figura 22 – Resultados da abordagem 4, marcadores e regiões segmentadas.

4.1.5 Casamento com template

Esse tipo de abordagem é através da convolução de um template, isto é, de uma imagem do objeto que se quer encontrar com a imagem onde o objeto se encontra. Assim o template desliza sobre a máscara efetuando cálculos de similaridade ou dissimilaridade. Quando o template passa pelo objeto o valor da similaridade é máximo ou no caso de abordagens de dissimilaridade esse valor é mínimo. Assim, após a convolução o casamento de template se resume a um problema de mínimos e máximos.

O casamento de template implementado foi feito utilizando templates de placas brasileiras encontrados na internet e numa segunda tentativa com imagem de placa recortada do banco de imagens. Apesar das tentativas de aproximar o template com a imagem por meio da aplicação de filtros ao template, essa técnica não obteve bons resultados para esse banco de dados por conta da variações de escala e luminosidade presentes nas placas.

Para corrigir aspectos como rotação e escala buscou-se a utilização de técnicas de localização de pontos chaves e descritores como o Orb, Sift e Surf. Entretanto a implementação dessas técnicas não foi bem sucedida com as imagens testadas e o algoritmo não evoluiu.

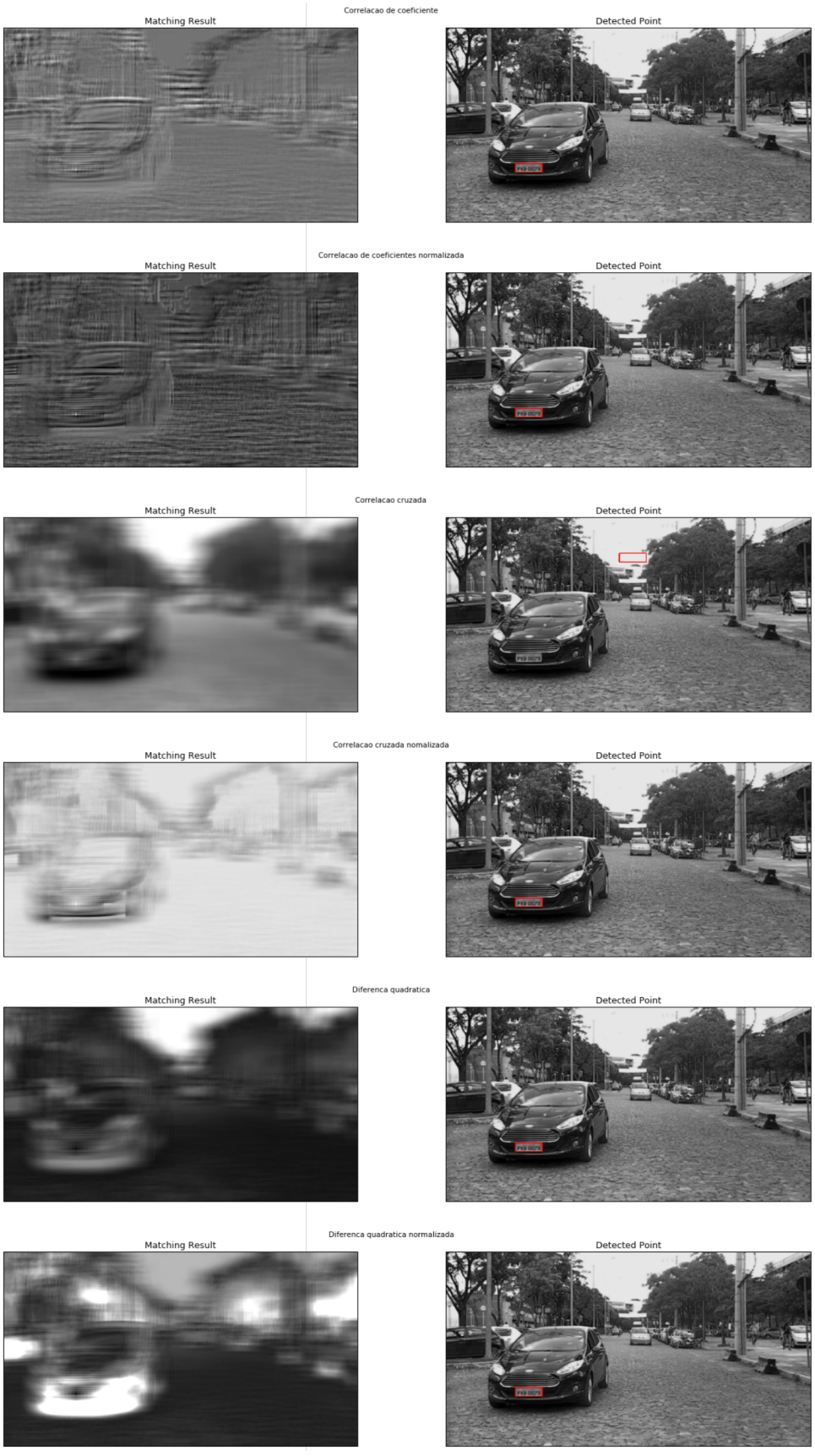
Em relação ao *template matching* com o uso da função *cv2.matchTemplate()*foram realizados testes utilizando quatro métricas de similaridade e duas de dissimilaridade entre imagens. As métricas utilizadas são:

- Correlação dos coeficientes (Similaridade)
- Correlação dos coeficientes Normalizada (Similaridade)
- Correlação Cruzada (Similaridade)
- Correlação Cruzada Normalizada (Similaridade)
- Diferença Quadrática (Dissimilaridade)
- Diferença Quadrática Normalizada (Dissimilaridade)

Na Figura 23 é exibido os resultados para cada uma das métricas apresentadas. Essas Figuras foram feitas com a utilização da própria placa do veículo como ferramenta de validação do algoritmo. As imagens a esquerda representam as saídas da função de casamento com template, já as imagens a direita as marcações em vermelho indicam as regiões onde foi detectado o template na imagem.

No caso das placas template retiradas do Google Imagens, tentou-se uma abordagem de template matching entre as imagens originais, binarizadas, filtradas e escalonadas.

Figura 23 – Validação do algoritmo de *template matching* com uso de template extraído da própria figura.



Nenhum dos testes feitos foi capaz de localizar a placa. Apesar de falhar nessa implemetação, essa técnica se usada corretamente pode solucionar diversos problemas. Neste trabalho ela ainda será explorada em outras situações.

4.2 Abordagens complementares para seleção do melhor candidato

Essa etapa do algoritmo é fundamental na complementação dos algoritmos listados acima. Entretanto uma vez que os algoritmos demonstrados acima tenham corretamente selecionado a placa existente, essa etapa atua de forma independente da anterior.

De modo geral, essa etapa serve para identificar imagens de placas de imagens que não são placas. As abordagens demonstradas neste trabalho foram divididas em dois grupos: Com uso de template e sem uso de template.

A abordagem com uso de template consiste de uma abordagem de comparação de histogramas. Enquanto que as abordagens sem template utilizam-se de operadores morfológicos e rastreamento de bordas para localizar o retângulo que define a placa ou para segmentar parcialmente os caracteres e contar o número de caracteres presentes.

4.2.1 Comparação de histogramas

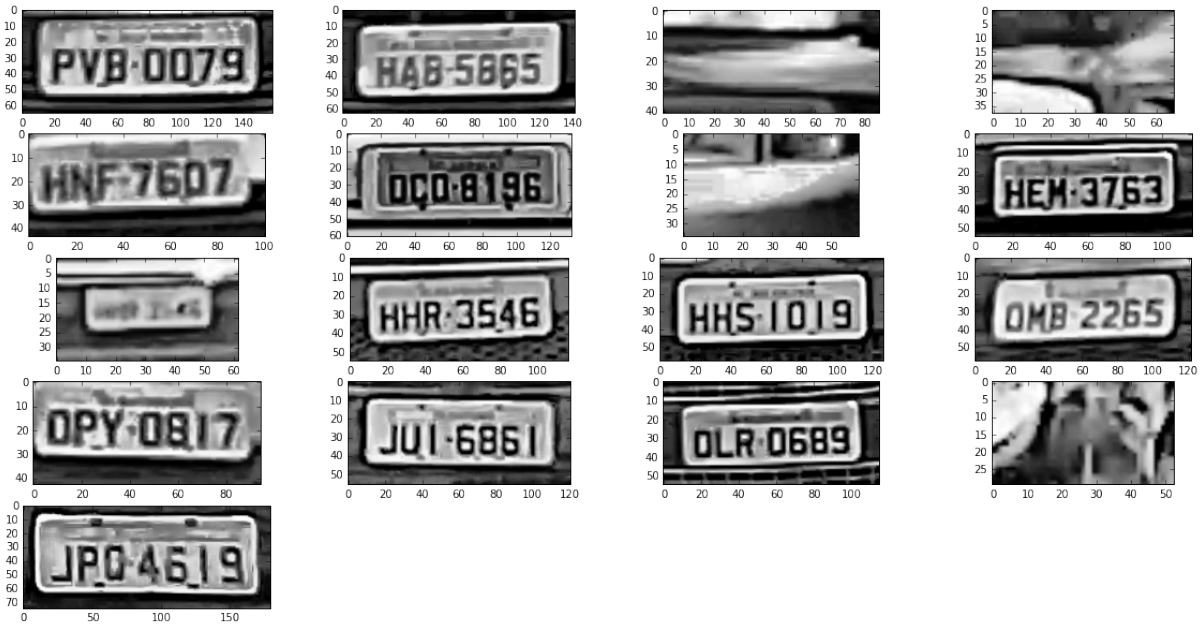
Nessa abordagem escolhe-se uma imagem de placa de carro template e calcula-se o seu histograma. Do outro lado calcula-se os histogramas das imagens das quais se quer selecionar apenas as placas. Essa métrica começa inicialmente equalizando os histogramas de todas as placas de forma automática de forma a homogeneizá-los e aumentar a semelhança entre histogramas de um mesmo objeto. Na Figura 24 pode-se visualizar um grupo de candidatos a placa, o objetivo desse algoritmo é remover as imagens que não representam placas.

A etapa inicial começa calculando os histogramas utilizando a função do Numpy *np.histogram()*, em seguida, esse histograma são equalizados pela função *cv2.equalizeHist()* . Na Figura 25 estão representados os histogramas normalizados das dezessete imagens ilustradas na Figura 24.

Escolhendo-se a primeira imagem como template, realizou-se a comparação dos histogramas por meio da função *cv2.compareHist()* . O OpenCV possui seis métricas de comparação de histograma já implementadas. Foram utilizadas todas as seis de modo a avaliar aquela ou aquelas que obtiveram os melhores resultados. As métricas utilizadas foram:

- Correlação
- Chi-quadrado,

Figura 24 – Conjunto de candidatos a placa para validação dos algoritmos de seleção do melhor candidato

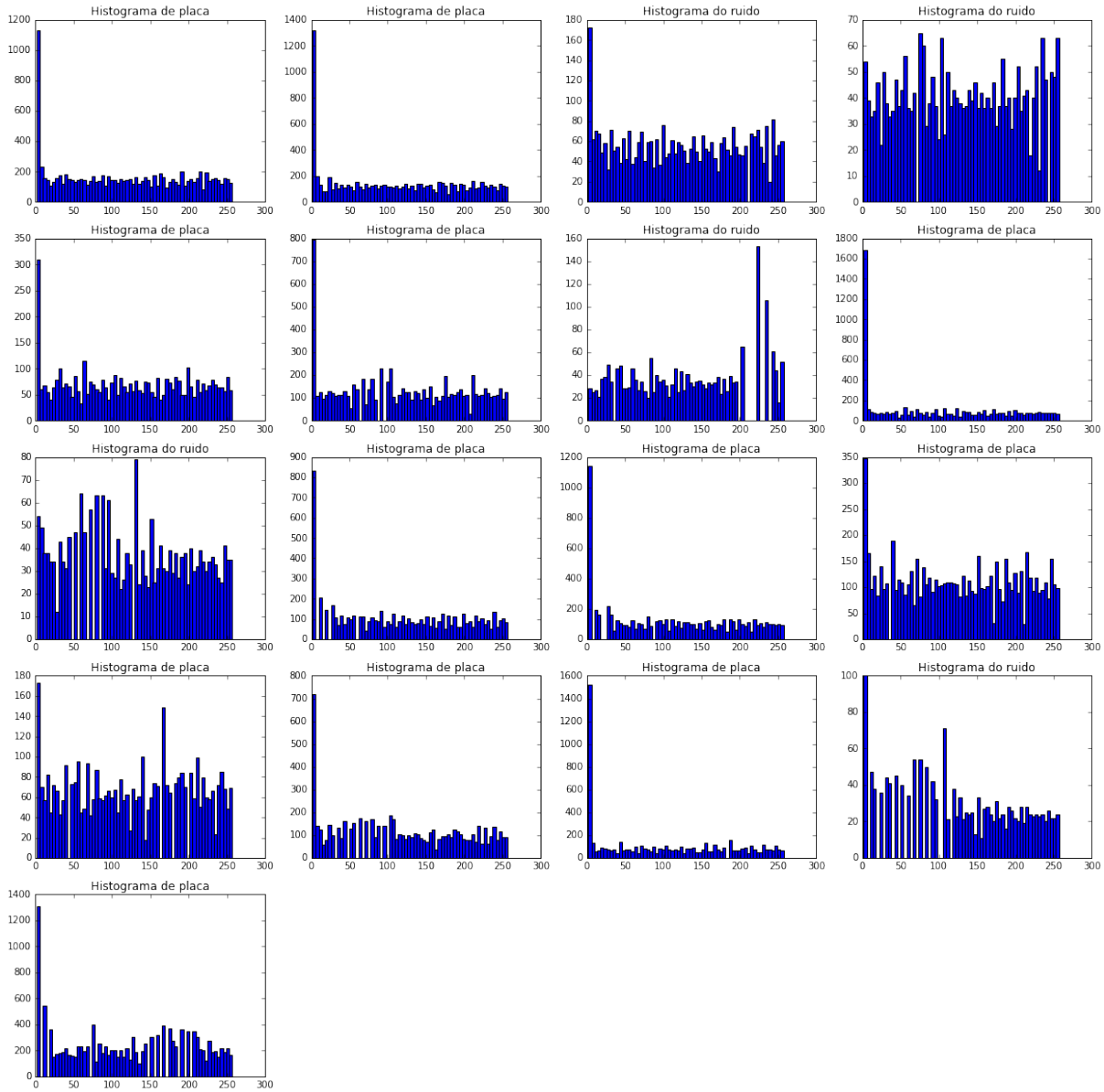


- Interseção,
- Bhattacharyya,
- Chi-quadrado alternativo
- Divergência de Kullback-Leibler

Na Tabela 2 a seguir observar-se os resultados dos scores obtidos na comparação de histogramas. Nessa tabela as imagens são nomeadas de 0 a 16 fazendo-se a leitura da esquerda para a direita e de cima para baixo. A imagem 0 representa o *template* e é demonstrado os valores de comparação dele com ele mesmo. As métricas de correlação e de Interseção são as única métricas de similaridade, isso significa que quanto mais parecidos os histogramas maiores serão os scores. Na Figura 26 é exibido o algoritmo utilizado para comparar os histogramas e gerar os valores da Tabela 2.

Da análise da tabela observou-se que o método da interseção e de Chi quadrado alternativo obtiveram os melhores resultados, isto é existe um valor de limiar que divide as imagens das placas das que não são placas. Para o método da interseção o limiar escolhido foi o de 3600 e o score das placas válidas deve ser maior do que esse valor. Já para Chi quadrado alternativo o limiar foi de 7000 e as imagens de placas devem possuir score menor que esse valor. Na Figura 27 está ilustrado as placas selecionados por ambos os métodos a partir do grupo de imagens da Figura 24.

Figura 25 – Histogramas equalizado dos candidatos



4.2.2 Localização do retângulo externo da placa

Essa abordagem e a posterior são bem similares e se baseiam no uso de dos rastreamento de contornos para identificar características da placa. Nessa abordagem buscou-se encontrar um retângulo na imagem. A Figura 28 mostra 45 candidatos a placa encontrados na execução da primeira abordagem de detecção apresentada. Destas 45 imagens apenas 14 são placas, essas imagens foram utilizadas para avaliar essa abordagem e como também a próxima abordagem.

A localização do retângulo externo da placa consiste da aplicação de uma limiarização por Otsu na imagem dos candidatos a placa, isto é, encontrado o *bouding box* de um possível candidato recorta-se a imagem na posição e dimensões desse *bouding box*.

Figura 26 – Algoritmo de comparação de histogramas.

```
#métodos de comparação de histograma
methods = [cv2.HISTCMP_CORREL,cv2.HISTCMP_CHISQR,cv2.HISTCMP_INTERSECT,cv2.HISTCMP_BHATTACHARYYA, cv2.HISTCMP_CHISQR_ALT,
           cv2.HISTCMP_KL_DIV]
# definição do histograma template
temp = hists[0].ravel().astype('float32')
# declaração do dicionário comparation, onde serão armazenados as listas com os resultados da comparação
comparation={
                "method": [],
                "score": [],
                "hist": []
            }
# loop para comparar o histograma template com os histogramas calculados anteriormente e armazenados na lista hists
i = 0;
for h in hists:
    #loop para a execução da comparação em todos os métodos
    for compare_method in methods:
        # função de comparação de histograma
        results = cv2.compareHist( temp, h.ravel().astype('float32'), compare_method )
        # preenchimento do dicionário
        comparation['method'].append(compare_method)
        comparation['score'].append(results)
        comparation['hist'].append(i)
    i = i+1
```

Tabela 2 – Scores obtidos com seis métricas de comparação de histograma para a abordagem um dos algoritmos de escolha do melhor candidato.

		Métricas de Comparação					
		Correlação	Chi-quadrado	Interseção	Bhattacharyya	Chi-quadrado alternativo	Divergência de Kullback-Leibler
Imagens	0	1.0	0.0	10270.0	0.0	0.0	0.0
	1	9,70E+15	6,95E+15	8505.0	9,34E+15	8,32E+15	1,79E+15
	2	7,14E+15	4,60E+13	3526.0	1,56E+15	7,19E+15	1,58E+16
	3	1,72E+16	5,94E+15	2546.0	2,06E+16	9,89E+15	2,24E+16
	4	8,66E+15	3,51E+16	4444.0	1,15E+14	5,15E+16	9,15E+15
	5	8,56E+15	1,62E+16	7557.0	1,95E+16	2,27E+16	1,60E+16
	6	9,59E+14	6,76E+15	2100.0	3,37E+15	1,19E+16	5,10E+14
	7	9,73E+15	2,73E+15	5804.0	1,78E+16	3,70E+15	6,16E+15
	8	1,90E+15	6,54E+15	2170.0	2,69E+16	1,12E+16	3,83E+16
	9	8,94E+15	2,25E+16	6264.0	2,16E+16	3,44E+16	2,28E+16
	10	9,30E+14	1,80E+16	7072.0	2,13E+15	2,80E+15	2,14E+16
	11	7,20E+15	1,67E+16	6890.0	1,49E+16	2,35E+15	7,77E+15
	12	4,48E+15	4,16E+13	4085.0	2,38E+13	6,67E+15	2,70E+15
	13	8,35E+15	2,32E+15	6470.0	2,55E+15	3,72E+16	3,16E+16
	14	9,73E+15	2,51E+16	5900.0	1,79E+16	3,53E+15	9,58E+15
	15	4,71E+16	7,47E+15	1590.0	3,57E+16	1,34E+16	7,05E+15
	16	7,48E+15	7,47E+14	8651.0	3,03E+15	6,40E+15	3,85E+16

Após a limiarização, aplica-se a funda de rastreamento de contornos configurada para rastrear somente os contornos mais externos, no *cv2.findContours()* utiliza-se o parâmetro *cv2.RETR_EXTERNAL*.

Esses contornos são aproximados a polígonos e os quadriláteros são selecionados. Aplica-se uma filtragem por Aspect Ratio e considera uma altura mínima para a imagem. O candidato que apresentar um contorno que preencha os requisitos acima é classificado como placa. Na Figura 29 é apresentado os candidatos aprovados por esse método. Observa-se que ele permitiu a passagem de algumas falsas placas, entretanto identificou todas as 14 placas e eliminou 28 falsos positivos.

Esse método apresentou uma vantagem extra que foi o aprimoramento da detecção da placa. Nesse caso, após aplicar essa técnica a caixa de marcação de algumas placas ficaram mais precisas, isto é, mais próximas da dimensão da placa. Na Figura 30 está

Figura 27 – Placas selecionadas pelo algoritmo de comparação de histogramas.

Figura 28 – Candidatos a placa para validação das abordagens de seleção de candidatos 2 e 3.

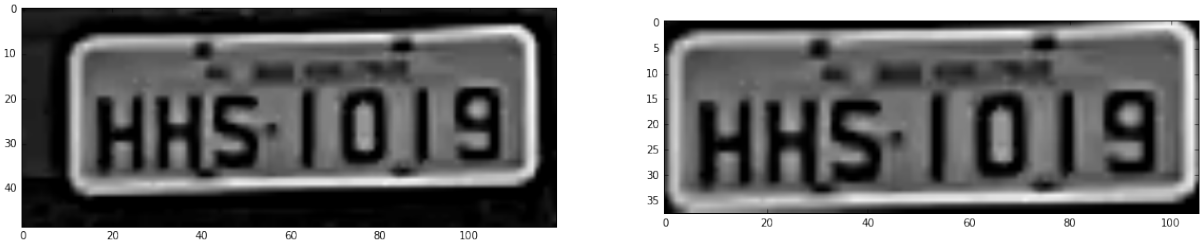
Figura 29 – Candidatos selecionados pela abordagem 2.

exemplificado esse efeito destacando-se uma placa antes e depois do método.

4.2.3 Contagem do número de elementos internos a placa

Essa abordagem é bem similar a anterior com a diferença que ao invés de buscar os contornos mais externos ela busca os contornos internos a ele. Para isso a configuração de contornos é configurada para encontrar todos os contornos e montar uma árvore hierárquica deles, através do parâmetro *cv2.RETR_TREE*. Em seguida, selecionam-se apenas os contornos que possuem ao menos um contorno pai, isto é, ao menos um contorno

Figura 30 – Aperfeiçoamento da *bouding box* pela abordagem 2.



externo a ele.

Então, determina-se um bouding box para esse contorno e realiza-se uma filtragem baseando-se nas dimensões mínima para a altura e numa dimensão máxima para o comprimento do bouding box. Para cada candidato conta-se o número de contornos dentro dessas características, se esse número for maior ou igual a quatro o candidato é selecionado como placa. Não escolheu-se um número precisamente igual à quantidade de caracteres presentes na placa devido a presença de outros elementos na placa mas principalmente pelo fato do algoritmo não realizar uma segmentação precisa desses caracteres na placa, o valor quatro foi obtido observando um grupo de placas e anotando a quantidade de regiões formadas nelas. Na Figura 31 observa-se os resultados obtidos para essa abordagem, nota-se que ela obteve 100% de acerto, eliminando todas as falsas placas e mantendo as 14 placas verdadeiras.

Figura 31 – Candidatos selecionados pela abordagem 3.

Como essa abordagem apresentou melhores resultados que o anterior, acrescentou-se uma segunda etapa a ela para fazer com que o bônus extra de melhoria do *bounding box* também estivesse presente. Isso foi feito aplicando o resultado dessa abordagem na abordagem anterior.

4.3 Técnicas de segmentação de caracteres

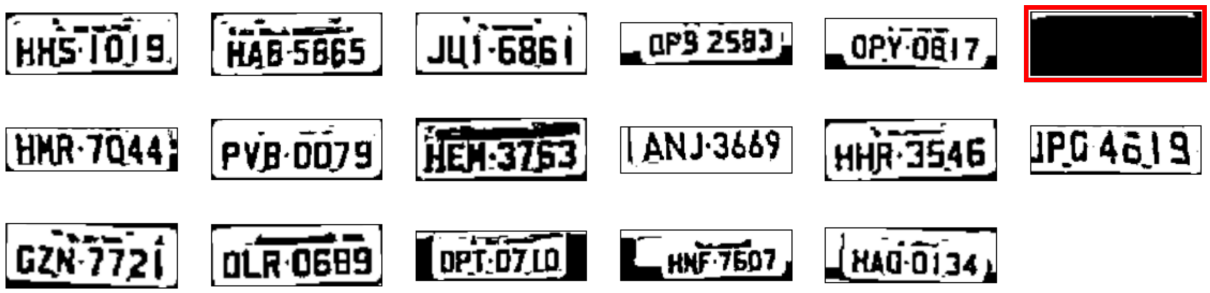
Essa técnica foi uma das mais complexas de se implementar devido às variações existentes nas placas. Variações de fontes, espaçamento entre os caracteres, diferentes resoluções e diferentes iluminações prejudicaram as abordagens testadas.

Neste trabalho foram testados alguns métodos, incluindo técnicas simples como dividir a imagem em porções iguais. As abordagens que resultaram nas melhores segmentações foram as que utilizaram a operação morfológica de fechamento na imagem binarizada da placa. Na Figura 32 está exibido a continuação do algoritmo apresentado no tópico anterior e nela observa-se as placas cujos caracteres serão segmentados.

Figura 32 – Placas para validação do algoritmo de segmentação de caracteres.

A primeira etapa desse algoritmo consistia na limiarização das placas, para isso foi utilizado o limiar obtido pelo método de Otsu. Entretanto, devido a irregularidade na iluminação das placas o método de Otsu falhou na limiarização de uma das placas acima, como pode ser visto na Figura 33 em que a placa falha aparece com o destaque vermelho.

Figura 33 – Limiarização por Otsu com falha na segmentação de uma das placas.



Para contornar esse efeito fez-se uso de um filtro de aguçamento que não é classificado como detector de bordas, o *unsharp mask*. Esse filtro consiste da subtração entre a imagem original e uma imagem borrada obtida a partir dela. Esse tipo de filtro destaca as regiões de transição, isto é as bordas, mas mantém os demais elementos da imagem. Na implementação feira realizou-se uma soma ponderada entre a imagem original da placa e essa mesma imagem após a aplicação de um filtro de suavização gaussiano. Os resultados da aplicação desse filtro são sutis e de difícil percepção, porém o aumento do contraste ocasionado por ele, corrige a falha gerada pelo método de Otsu.

Observa-se que a saída da limiarização por Otsu gera imagens de fundo branco com os caracteres em preto. A fim de reduzir as conexões de um caractere com caractere e entre caractere e as partes escuras presentes na placa e fora dela, optou-se por utilizar o operador morfológico de fechamento. Como já discutido em outras seções, esse operado busca unificar

regiões brancas e com isso acaba por eliminar pequenas conexões em preto. Após isso, realiza-se o rastreamento de contornos seguido de filtragem por tamanho mínimo de altura e máximo de largura, como também de AR para garantir que as caixas de marcação serão retângulos na vertical. Na Figura 34 visualiza-se os caracteres segmentados através desse método, como também observa-se que a placa a qual o método de Otsu havia falhado previamente pode ser interpretada.

Figura 34 – Segmentação de caracteres obtida após a aplicação do filtro morfológico de Fechamento.

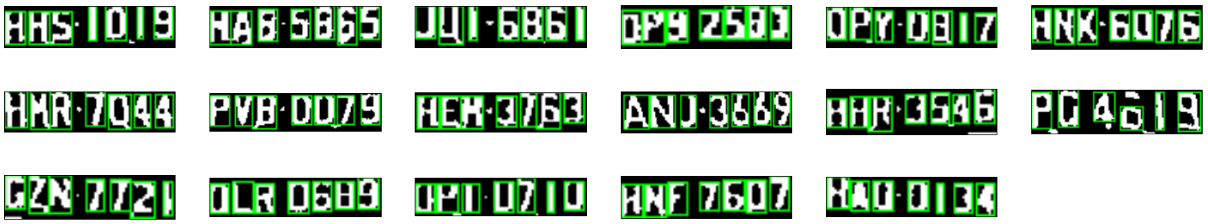
Observou-se que esses resultados apresentam pequenas inconsistências como o fato de algumas marcações cortarem a letra na metade, ou a ocorrência de um excesso de marcações para um mesmo caractere ou uma única marcação para dois caracteres ou ainda a não marcação de caracteres. Decidiu-se então aplicar uma segunda etapa a esse algoritmo para tentar aperfeiçoá-lo.

A segunda etapa utiliza o resultado da primeira para reduzir os objetos indesejados na imagem. Para isso, converte a anotação das caixas delimitadoras calculadas da forma x,y,comprimento e largura para a forma x_1,y_1,x_2,y_2, em que (x_1, y_1) formam o ponto superior esquerdo da caixa e (x_2, y_2) o ponto inferior direito. O algoritmo encontra então o ponto (x_1, y_1) mais próximo da origem e o ponto (x_2, y_2) mais distante dela. Com os pontos mínimo e máximo recorta a imagem original da placa com base nessas coordenadas.

Em seguida, segmenta-se a imagem, novamente, com a técnica de Otsu e aplica se uma erosão com um elemento estruturante pequeno e em forma de cruz para tentar dividir os caracteres que ficaram agrupados. Realiza-se então o rastreamento de contornos e filtra-se o contorno com base na largura e AR mínimos e na largura máxima. Se o AR for menor que 1 (no algoritmo considerou 0,95) significa que o comprimento do delimitador é maior que sua altura, como para um único caractere a altura deve ser a maior dimensão, esse AR indica que esse delimitador está demarcando ao menos dois caracteres. Fazendo-se essa suposição o algoritmo simplesmente divide esse marcador pela metade, gerando dois marcadores um para cada provável caractere. Os resultados dessa segmentação estão ilustrados na Figura 35.

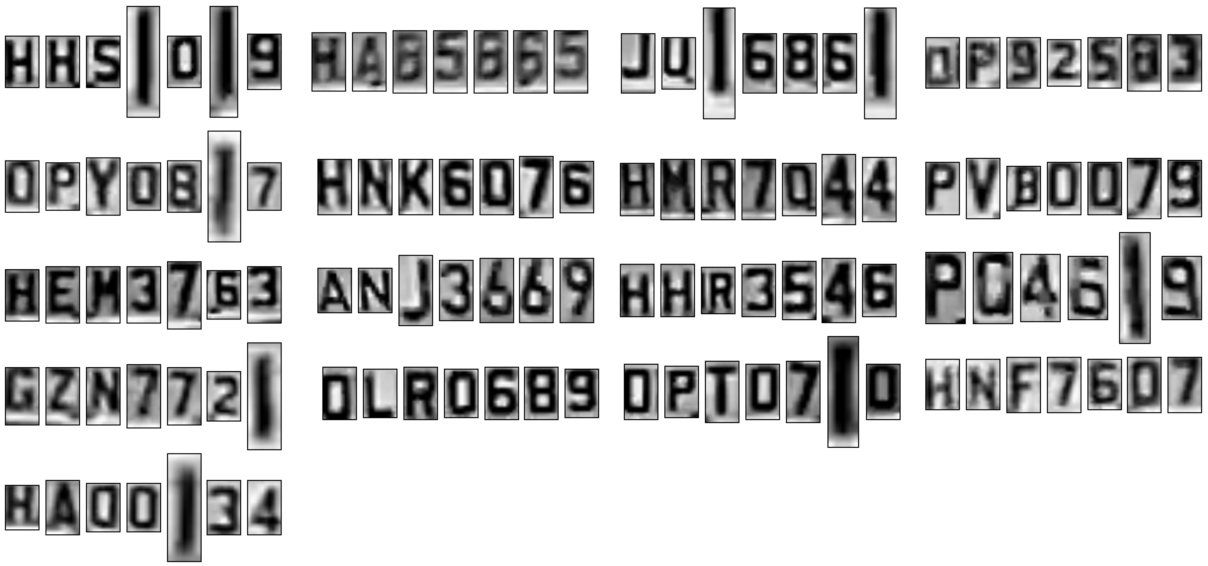
Criou-se ainda uma rotina para corrigir as caixas delimitadoras quando essas não capturam o caractere por inteiro, considerando-se apenas o eixo vertical. Essa rotina

Figura 35 – Segmentação de caracteres com algoritmo em duas etapas.



captura o caractere com maior altura e compara todos os outros em relação a esse, caso haja um caractere cuja altura do delimitador é menor que uma porcentagem da altura máxima, a altura desse caractere passa a ser igual a máxima. Se isso acontecer, a rotina procura verificar se essa *bounding box* está localizada muito abaixo do padrão das demais e caso esteja altera o valor de y desse delimitador para o valor mínimo. Finalmente, a abordagem ordenada em ordem crescente os caracteres encontrados com base no valor do x dos delimitadores. Dessa forma, os recortes da placa podem ser feitos na ordem de leitura da placa. Na Figura 36 estão presentes os recortes dos caracteres ordenados para o reconhecimento das placas exibidas na Figura 32. Observa-se na imagem da décima segunda placa (JPG-4619) o efeito da rotina de correção do delimitador sobre o eixo da vertical nos caracteres 4 e 6 cujo recorte considerou todo o caractere, embora suas *bouding boxes* haviam delimitado apenas partes deles. Apesar dessa correção a placa citada continuo como sendo uma falha do algoritmo pois não consegui detectar a letra J, assim das 17 placas, 16 foram corretamente segmentadas.

Figura 36 – Recorte dos caracteres segmentados.



4.4 Algoritmos para Reconhecimento Óptico de Caracteres (OCR)

A última etapa do reconhecimento automático de placas veiculares é a leitura dos caracteres da placa. Diversas técnicas vêm sendo estudadas para melhorar os atuais resultados obtidos. Em geral, utilizam-se modelos de aprendizado de máquinas para a realização dessa tarefa. Dos desafios atuais podem ser citados a adaptabilidade a múltiplas fonte e a leitura de placas em imagens de carros em alta velocidade ou capturadas por câmeras de baixa resolução. Assim, pesquisadores buscam alternativas para otimizar essa etapa, dentre as soluções propostas encontram-se métodos para seleção do melhor frame e o uso da técnica de super resolução.

Em todo caso, pode-se fazer uso de engines de OCR públicas como é o caso do Tesseract desenvolvido por pesquisadores da Google. Essa engine possibilita o retreinamento de suas redes para adequação a novos cenários. A plataforma OpenAlpr utiliza o Tesseract para o reconhecimento de placas e disponibiliza seu código e APIs para uso público. Essa plataforma realiza todas as etapas de um algoritmo de reconhecimento de placas e possui um modelo treinado para placas brasileiras.

Neste trabalho desenvolveu-se duas soluções para o desafio do OCR de placas veiculares em ambiente externo. As abordagens escolhidas foram o uso de *template matching* ou casamento de templates e a criação de uma rede neural artificial simples. De modo geral, o foco não foi a obtenção dos melhores resultados, mas sim no desenvolvimento da solução, servindo de guia para futuros trabalhos.

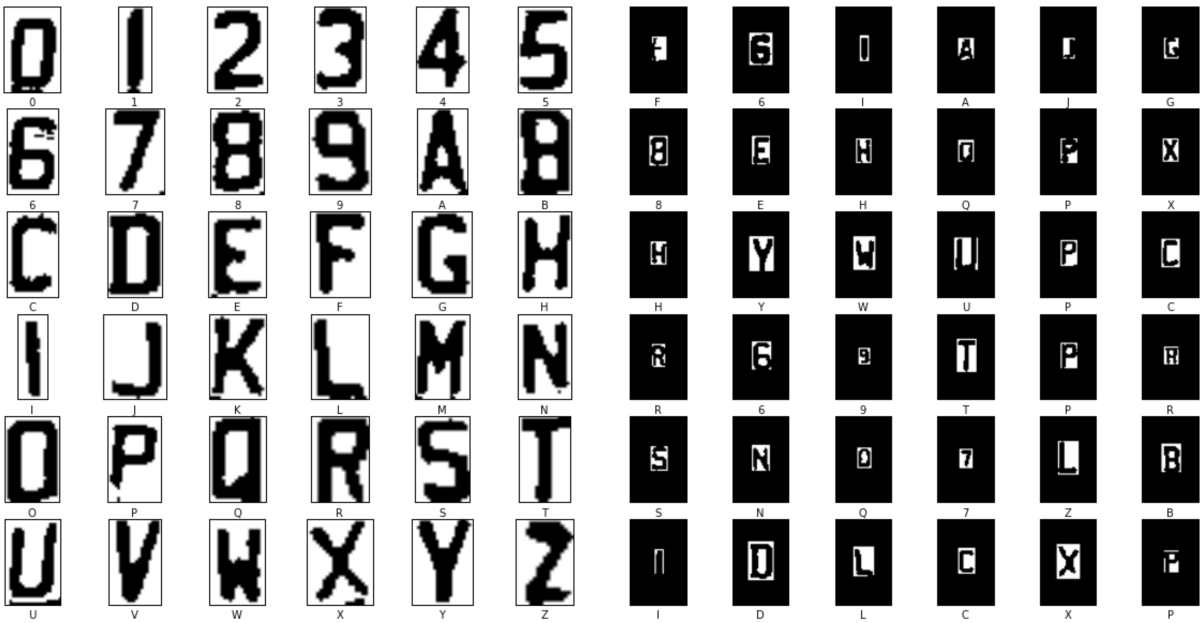
4.4.1 *Template Matching*

A abordagem de template matching para OCR consiste em dada uma imagem de uma letra ou números, comparar essa imagem com templates de todas as letras e números e identificar aquele que obteve o melhor score. Como já explicado o Template Matching funciona por meio da convolução do template sobre a imagem. Durante essa convolução são realizados cálculos matemáticos e estatístico de similaridade e dissimilaridade. Para a detecção a posição da imagem que apresentou o melhor score é a provável posição do objeto na imagem. Para o reconhecimento, utiliza-se o template matching como método de comparação entre imagens de modo a determinar se as imagens estão correlacionadas ou não.

A metodologia adotada para essa abordagem consistiu num primeiro momento em gerar templates a partir das imagens do banco de dados. Criou-se então um código em Python que carrega imagens e suas anotações do banco e recortava os caracteres da placa com base nos valores indicados na anotação. Essas imagens de caracteres foram então salvas. A partir de então criou-se duas rotinas para preparação das imagens para o casamento de templates.

A primeira rotina carrega e prepara os arquivos templates, isto é, ler a imagem e aplica a limiarização por Otsu e deixando o resultado com fundo branco e letra em preto. A segunda rotina foi inicialmente desenvolvida para a abordagem com uso de RNA, porém também foi aproveitada nesse método. Ela serve para preparar os dados a serem comparados com o template. Dada uma imagem, a rotina a aplica a limiarização tentando manter o fundo branco com caractere em preto. Em seguida, deixa a imagem com uma resolução espacial fixa a partir da criação de uma estampa de zeros ao redor da imagem. Na Figura 37(a) são apresentados algumas imagens utilizadas como template, enquanto que na Figura 37(b) imagens a serem comparadas já com a estampa de dimensões fixas.

Figura 37 – Imagens utilizadas para realização do *Template Matching*.



(a) Templates (b) Imagens para teste

Devido a essa estampa aplicada a imagem teste, o template possui dimensões menores que a imagem teste facilitando o casamento. Com essas imagens foram realizados testes utilizando-se todas as métricas de comparação de similaridade e dissimilaridade de imagens implementadas na função *cv2.matchTemplate()*. Nesse teste foi executado o *template matching* sobre as imagens testes com uso de um único template por caractere. Ao todo foram utilizado um total de 140 imagens nos testes preliminares.

Nos testes, o algoritmo reconheceu corretamente 33 das 140 por pelo menos uma das métricas de comparação. Nessa abordagem a técnica de comparação que mais acertou foi a Correlação Cruzada normalizada acertando 20 das 140 imagens.

4.4.2 Redes Neural Artificial

As abordagens mais comuns para realização de um OCR envolvem os trabalhos com aprendizado de máquinas. No campo da visão computacional o método de aprendizado de máquinas mais utilizado são as redes neurais artificiais, como as redes convolucionais. Neste trabalho foi desenvolvido uma abordagem simples para realização de um OCR com uso de redes neurais. Foi utilizado a biblioteca de *machine learning* e *deep learning* TensorFlow em linguagem Python com o auxílio da biblioteca Keras. O Keras é uma biblioteca que utiliza o TensorFlow (ou o Theano) como *backend* para aplicações em redes neurais.

A rede implementada nesta abordagem não constitui uma rede profunda, mas foi suficiente para obter bons resultados para o OCR no contexto do reconhecimento automático de placas veiculares. Além disso, o objetivo maior de apresentar essa abordagem é demonstrar um caminho para iniciação na programação de redes neurais artificiais. Essa seção será dividida em tópicos para deixar mais claro o passo a passo do algoritmo.

4.4.2.1 Criação de uma base de imagens para OCR

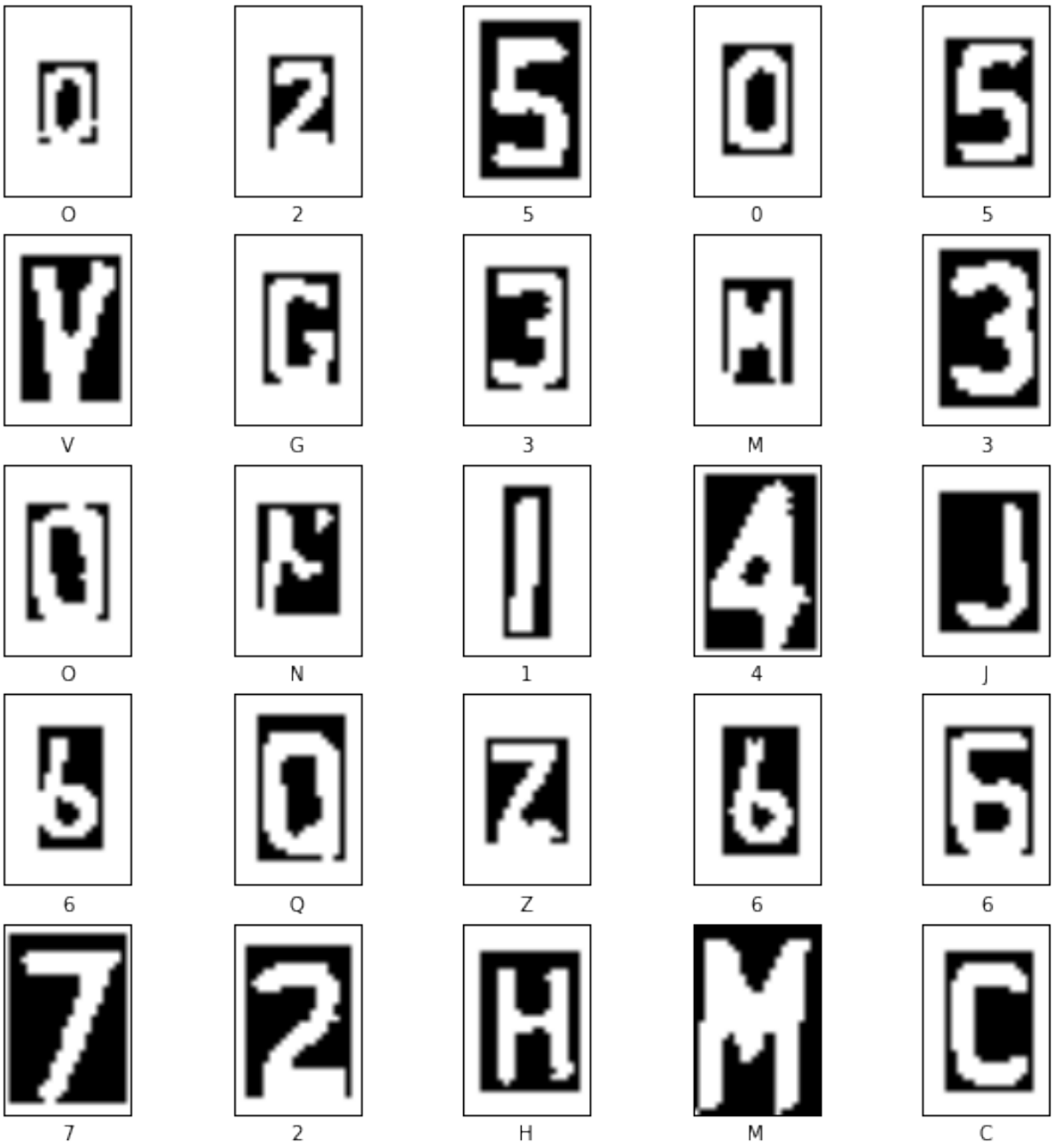
Para criar a base de imagens para o OCR programou-se dois códigos em Python, uma para segmentar os caracteres das imagens com base nos valores anotados na base e outro para binarizar a imagem tentando manter um padrão comum para todas elas (fundo branco e caracteres em preto). Normalmente, em redes neurais as entradas da rede são vetores ou matrizes de mesma dimensão, entretanto, os recorte dos caracteres feitos pelo primeiro código geraram imagens de diferentes resoluções espaciais. Assim optou-se por aplicar a essas imagens binarizadas no segundo código uma estampa que iguala-se as resoluções. Além disso, o TensorFlow Keras utiliza como padrão de entrada de suas funções um vetor de três dimensões dados pela profundidade, altura (número de linhas) e comprimento (número de colunas), então o segundo algoritmo acrescentou também o valor da terceira dimensão as imagens de entrada da rede.

Durante o algoritmo um tomou-se o cuidado de acrescentar ao nome das imagens a serem salvas na base o valor que ela representava, isto é, o número ou letra indicado por ela. Isso foi feito de forma automática extraindo esse valor dos valores de anotação da base durante a execução do primeiro algoritmo. Essa implementação facilitou a extração do "labels" do conjunto de imagens de treinamento, testes e validação. Na Figura 38 exibe-se algumas imagens do banco de dados criado, a resolução dessas imagens foi definida como sendo 22px de largura por 33 px de altura. Ao todo foram criadas 5628 imagens para o treinamento e 5523 imagens para o teste, numa proporção de aproximadamente 50%.

4.4.2.2 Preparação dos dados no algoritmo

Para essa abordagem foram criadas diversas funções para armazenar em listas e vetores as imagens e labels a partir da informação do caminho do diretório onde se

Figura 38 – Imagens geradas para utilização em RNA.



encontram. Utilizou-se bibliotecas do Python como o glob e glob2 para realizar uma busca interativa nas pastas presentes em um diretório.

Como as imagens foram salvas com três canais, essa etapa é responsável por fazer a conversão da imagem de três canais para um único canal. Para os labels utilizou-se um dicionário para demonstrar uma forma de converter rótulos textuais em representações numéricas. Para o problema do OCR, essa etapa poderia ser feita com a conversão direta de texto para inteiro, uma vez que o rótulo é o próprio número ou letra que ele representa.

O código se inicia carregando todos o caminhos de arquivo das imagens de treina-

mento e de teste e armazenando-os em listas. Utiliza-se, em seguida, a função do Numpy *np.random.shuffle()* nas listas para misturar os nomes dos arquivos. Verifica-se se o procedimento foi bem executado ilustrando-se algumas imagens de treinamento e teste e escrevendo na tela o valor dos *labels* correspondentes.

As imagens de treinamento e teste passam por uma etapa de normalização a partir de sua divisão com seu valor máximo (255), obtendo-se os vetores de treinamento e teste normalizados entre 0 e 1. Ao final dessa etapa os dados ficam prontos para a aplicação no modelo.

4.4.2.3 Definição do modelo

De forma geral, foi criado um modelo sequencial, com três camadas, sendo uma camada de entrada *flatten*, uma camada densa como 256 neurônios e função de ativação *nn.relu* e outra camada densa de saída com 36 neurônios, igual ao número de labels, e função de ativação *softmax*. A seguir serão explicados cada um desses termos.

O Keras é uma biblioteca cujo núcleo de suas operações são os modelos de uma rede. O tipo de modelo mais simples é o modelo sequencial que consiste de uma sequência linear de camadas. A primeira etapa na construção do modelo é informar a ele o tamanho da entrada esperada. Isso é feito na definição da primeira camada, não sendo necessário para as camadas subsequentes, através do parâmetro *input_shape*. Esse parâmetro recebe uma tupla de valores que indicam o tamanho do elemento individual da entrada.

A camada *Flatten* transforma o vetor 2d de entrada em um vetor unidimensional, mudando o formato do dado para a entrada nas camadas de aprendizado. As camadas *Dense* constituem as camadas de RNA densamente conectadas. São camadas de neurônios do tipo perceptron cuja função de ativação é definida no parâmetro *activation*. São nessas camadas onde o aprendizado da rede realmente ocorre.

Na primeira camada de RNA do modelo proposto utilizou-se a função de ativação *ReLU (Rectified Linear Unit)* que é uma das funções de ativação mais utilizadas nas redes convolucionais e de *deep learning*. Da forma que foi implementada nessa abordagem o parâmetro *nn.relu* utiliza a forma padrão do retificador linear, isto é, uma função rampa iniciada a partir do valor zero (função identidade para valores maiores que 0). Assim essa função restringe os valores de saída dos neurônios para a faixa de 0 a infinito.

Na segunda camada densamente conectada a função de ativação proposta foi a função *Softmax*. Essa função é também conhecida como a função de ativação sigmoide ou logística para classificação em múltiplas classes. Sua principal característica é restringir o intervalo da saída do neurônio para valores entre 0 e 1, com isso é bastante utilizado em classificadores, pois sua saída indica a probabilidade de um determinado objeto pertencer àquela classe. A soma das saídas de uma camada com função de ativação *Softmax* deve ser

igual a um. Assim, fica claro perceber o motivo dessa camada ser a saída da rede proposta, como também, o motivo do seu número de nós ser igual ao número de *labels* do problema. Essa função é também definida através do parâmetro *activation=tf.nn.softmax*. Na Figura 39 observa-se a implementação do algoritmo de criação do modelo.

Figura 39 – Algoritmo de criação do modelo.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(33, 22)),
    keras.layers.Dense(256, activation=tf.nn.relu),
    keras.layers.Dense(36, activation=tf.nn.softmax)
])
```

Através do método *model.compile()* o parâmetros do treinamento são configurados para o modelo indicado. Essa função recebe três parâmetros principais: o otimizador, a função de perda e uma lista de métricas. O otimizador é a função responsável por alterar os pesos da rede durante a etapa de treinamento de modo que ela possa obter uma função de perda mínima. Em outras palavras, seja o erro entre a saída esperada da rede e a saída predita por ela a função de perda, o otimizador é o responsável por modelar os parâmetros da rede para que esse erro seja mínimo. O otimizador base para o estudo das RNAs é o gradiente descendente. O otimizador utilizado em nossa rede foi o Adam que é uma extensão do gradiente descendente estocástico com uso de *learning rate* adaptativo calculado a partir da média do segundo momento dos gradientes. Ele é bastante utilizado devido ao fato dele ser uma combinação dos otimizadores AdaGrad e RMSProp unindo suas melhores características, isto é, respectivamente bom em problemas de visão computacional e processamento de linguagem natural (gradientes esparsos) e bom em situações não estacionárias ou ruidosas. Além disso, é conhecido por alcançar bons resultados rapidamente.

A função de perda ou função de custo determina durante a etapa de treinamento o quão longe o modelo estar dos valores de saída esperado. Ela serve de guia para o otimizador, cuja função é minimizá-la. Na rede proposta, a função de perda utilizada foi a Entropia Cruzada Esparsa Categórica. Uso de entropia e entropia cruzada são fórmulas estatísticas poderosas para o cálculo de proximidade entre dois elementos. Esse método foi utilizado em diversas comparações ao longo deste trabalho, como exemplo é uma das métricas de comparação do *template matching* do OpenCV. O termo esparso serve para expandir o conceito dessa função uma vez que em RNAs o uso de entropia cruzada categórica é indicado para saídas com empacotamento binário (*one-hot enconde*) e a esparsa para saídas cujos valores são representados por números inteiros.

O último parâmetro do método *compile()* corresponde a uma lista de métricas para validação, avaliação e verificação de características do seu treinamento. Neste trabalho utilizou-se apenas o parâmetro de acurácia para medir o quão próximo as predições do modelo proposto na etapa de treinamento estão próximas dos valores dos descritos nos

labels de treinamento. Na Figura 40 está representado o algoritmo da implementação desse método.

Figura 40 – Algoritmo de compilação de modelo.

```
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

4.4.2.4 Preenchimento do modelo e treinamento

A próxima etapa consiste em preencher o modelo com os dados e os labels de treinamento. O Keras utiliza vetores numpy como parâmetros de entrada de dados e rótulos no método *model.fit()*. Além desses parâmetros a função *fit* permite determinar o número de épocas de treino até a parada, funções de *callback*, dados de validação, passos por época, proporção de divisão dos dados para validação, entre outros parâmetros relacionado ao treinamento da rede. Após a execução dessa função o treinamento da rede é iniciado. Por padrão (parâmetro *verbose*=1) essa função exibe um acompanhamento do treinamento mostrando os valores obtidos nas métricas especificadas no *model.compile()*. Neste trabalho, utilizou-se 5628 imagens para o treinamento e a classificação em 36 *labels* correspondentes as 26 letras do alfabeto e aos 10 números (0-9). Percebeu-se ainda que um total de 5 épocas foi suficiente para completar o treinamento com 95,59% de acurácia. Na Figura 41 está representado um dos treinamentos efetuados.

Figura 41 – Algoritmo de preenchimento e avaliação de modelo.

```
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
5628/5628 [==============================] - 1s 106us/step - loss: 0.1298 - acc: 0.9600
Epoch 2/5
5628/5628 [==============================] - 1s 100us/step - loss: 0.1089 - acc: 0.9655
Epoch 3/5
5628/5628 [==============================] - 1s 99us/step - loss: 0.0911 - acc: 0.9755
Epoch 4/5
5628/5628 [==============================] - 1s 99us/step - loss: 0.0857 - acc: 0.9717
Epoch 5/5
5628/5628 [==============================] - 1s 97us/step - loss: 0.0720 - acc: 0.9787

<tensorflow.python.keras.callbacks.History at 0x7f9542bf4890>
```

```
test_loss, test_acc = model.evaluate(test_images,test_labels) #(t2_images, t2_lab)

print('Test accuracy:', test_acc)
```

```
5523/5523 [==============================] - 0s 46us/step
('Test accuracy:', 0.8153177621613674)
```

4.4.2.5 Teste do modelo com as imagens e predição em novas imagens

Nessa etapa se utiliza o método *model.evaluate()* para testar o modelo e o treinamento. Essa função recebe como parâmetros os dados de teste e seus respectivos *labels* e calcula o resultado das funções de perda e das métricas definidas no *model.compile()*

para esse novo arranjo de dados. O modelo treinado obteve uma resposta de 80,52% de acurácia nos testes. Na Figura 41 também está representado uma das avaliações do modelo realizadas.

Para realizar predições gerou-se novas imagens a partir de imagens da base que não haviam sido utilizadas nem no treinamento e nem nos testes. Aplicou-se todas as etapas de preparação da imagem, isto é, recortou-se os caracteres da placa, salvou em disco com o nome contendo o caractere que ela representa, realizou sua binarização conforme o padrão adotado, aplicou a estampa para fixar a resolução e adicionou a terceira dimensão a imagem para a padronização do TensorFlow. Em seguida, utilizou-se o método *model.predict* para fazer a predição das imagens. Essa função recebe como parâmetro principal um vetor de dados com um único elemento ou um vetor com mais de um elemento. Para as 140 imagens testadas na abordagem de OCR anterior, a rede acertou 105 caracteres, ou seja, um percentual de 75%.

4.5 Detecção de veículos com uso de redes pré-treinadas

Uma das etapas de um sistema de ALPR para ambientes externos é a localização do veículo na imagem. A forma mais comum é a utilização de redes neurais de detecção e classificação. No campo das redes de detecção existem diversas redes que são estado na arte entre elas destacam-se *SSD (Single Shot Detection)*, *YOLO (You Only Look Once)* e *Faster-RCNN (Faster Region with Convolutional Neural Network).*

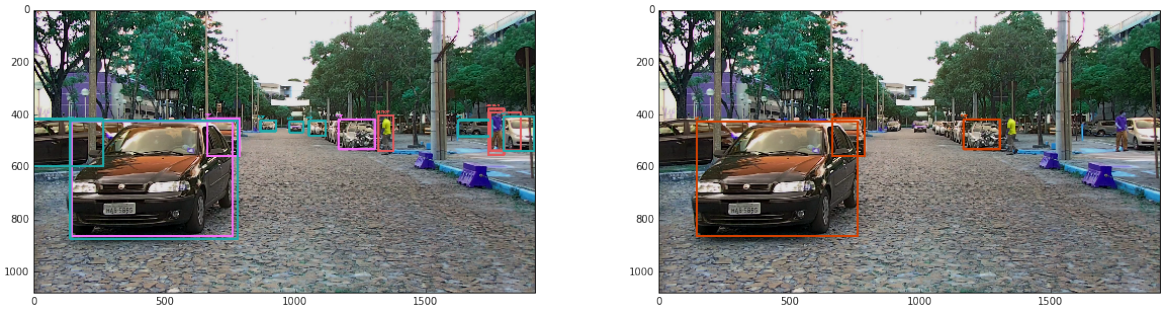
A tarefa de criação de uma arquitetura de rede que alcancem bons resultados como essas não é uma tarefa fácil. Entretanto boa parte delas possuem modelos pré-treinados que podem ser incorporados aos códigos através do TensorFlow Keras ou o módulo de Deep Learning do OpenCV ou ainda outras bibliotecas existentes. Neste trabalho, optou-se por utilizar as redes pré-treinadas para fazer uma demonstração sobre as redes de detecção de veículos. As redes usadas nessa demonstração foi a Yolov3 importada a partir do módulo DNN do OpenCV e a uma rede SSD com meta arquitetura em MobileNet incorporada a partir da API de detecção de objetos do TensorFlow.

Para usar o módulo de DNN do OpenCV, juntamente com Yolo, foi preciso realizar o download do arquivo de configuração da rede, um arquivo com as sua *labels* padrões e uma arquivo com seus pesos pré-treinados. Utiliza-se a função *cv2.dnn.readNet()* para carregar os pesos e a rede. Em seguida utiliza-se a função *cv2.dnn.blobFromImage()* para agrupar as imagens de entradas, de modo que a rede possa fazer a inferência em blocos. Por fim, utiliza-se o método do objeto "net", obtido após a leitura dos pesos e da rede, *net.forward()* para realizar as inferências ou predições sobre os grupos de imagens gerados, o resultado é um vetor com os valores de saída da rede. A próxima etapa é uma filtragem pelo *scores*, em que se seleciona apenas as predições com maior certeza de estarem corretas,

e então, desenha-se os *bounding boxes* sobre elas. Por fim, é necessário uma etapa de redução de múltiplas detecções para um mesmo objeto.

Devido aos métodos utilizados no Yolo, pode ocorrer dele selecionar múltiplas bounding boxes para um único objeto. Nesse caso utiliza-se uma algoritmo Non-max suppression, que realiza uma redução das caixas delimitadoras através da análise dos scores. Esse algoritmo foi aplicado com o módulo DNN na função *cv2.dnn.NMSBoxes()*. Na Figura 42, está exibido resultado da detecção de objetos feita pelo YOLOv3, na primeira tem-se todos os objetos encontrados e na segunda apenas os três primeiros objetos encontrados, o filtro da confidência dos scores utilizado foi de 0,5.

Figura 42 – Detecção de objetos com YOLOv3.



(a) Todos os objetos encontrados.

(b) Objetos com maior confiabilidade.

Para o uso da API de detecção de objetos do TensorFlow, o algoritmo é similar ao anterior, porém com algumas etapas mais complexas. Inicialmente faz-se o download do modelo, uma SSD MobileNET treinada com a base do COCO (*Common Object in Context*), e as *labels* dessa base de imagens. Utiliza-se o conjunto de métodos do objeto *tf.Graph()* para carregar o modelo. Em seguida, faz-se o mapeamento dos labels, isto é, indicar que o número 3 da saída de rede corresponde a classe "carro". Para isso, utilizou-se a função *label_map_util.create_category_index_from_labelmap()* do módulo de funções do módulo Python de utilidade (*from utils import label_map_util*). A próxima etapa é a preparação do gráfico tensorflow para a inferência, que possui diversas etapas como a formação de dicionário de tensores e adaptação das máscaras da rede para os tamanhos da imagem. Por fim a inferência é feita através do *tf.Session.run()*. Em seguida, é feita uma implementação que seleciona as imagens com melhores *scores*, a saída dessa rede já organiza os resultados com base no *score*, logo essa implementação apenas seleciona os n melhores resultados. Na Figura 43, está exibido resultado da detecção de objetos feita pela rede SSD MobileNet, na primeira tem-se todos os objetos encontrados e na segunda apenas com o objeto de maior *score*.

No próximo capítulo serão apresentados alguns resultados obtidos para os algoritmos que apresentaram um melhor desempenho nos testes iniciais. No caso dos algoritmos de detecção utilizou-se um código em *BASH* para executar o algoritmo em todas as 2000

Figura 43 – Detecção de objetos com SSD.

(a) Todos os objetos encontrados.

(b) Objeto com maior confiabilidade.

imagens da base. Para fazer isso, foi preciso usar o módulo *argparse* do Python.

5 Resultados

Nesta seção serão abordados alguns dos resultados obtidos com para os métodos que demonstraram mais estáveis. Entretanto, não se pode descartar a possibilidade de utilização desses algoritmos em outros cenários. Além disso, reforça-se o fato dos resultados obtidos não serem o real objetivo desse trabalho.

5.1 Algoritmos de localização de placas

Dentre os algoritmos de detecção foram calculados resultados para as abordagens 1 e 2. Essas abordagens foram combinadas com os algoritmos de seleção de melhor candidato a placa. os testes feitos variam alguns dos principais parâmetros das funções e foi executado a função sobre as duas mil imagens do banco de dados.

Para se avaliar as abordagens de detecção utilizou-se a métrica da razão entre a interseção e a união. Essa é um método muito utilizado para avaliar a eficácia de detectores baseados em caixas delimitadoras. Ela consiste em calcular a área da interseção entre a bounding box predita pelo algoritmo e a bounding box da anotação feita, considera-se essa como o resultado final a que se quer chegar, e a área da união dessas bounding boxes. Assim se elas não possuem intercessão o resultado foi errado e o IoU (Intersection over Union) resultará em zero. Caso as bounding boxes coincidam perfeitamente a área da união será igual a área da interseção e o resultado do IoU será um. Qualquer outra configuração resultará em valores entre zero e um, quando mais próximos de zero, significa que a caixa delimitadora predita está próxima da original, pois existe uma interseção mas ela não é uma boa marcação para o problema. Caso esteja próxima a um significa que a predição realizada chegou próximo do ideal. Neste trabalho, considerou utilizar três avaliações com base no IOU. Para uma avaliação mais binária, considerou-se que valores maiores ou iguais a 0,5 seriam considerados predições corretas e valores menores, incorretas. Entretanto, a forma mais comum de utilização do IoU elabora uma categorização das predições. Assim, neste trabalho dividiu-se os resultados em quatro categorias:

- Errado: quando o IoU é igual a zero;
- Ruim: quando o IoU é maior que zero e inferior a 70%;
- Boa: quando o IoU é maior que 70% e inferior a 90%;
- Excelente: quando o IoU é maior que 90%

Para a abordagem de detecção e localização número um, detecção de bordas com limiarização global e representação por rastreador de contornos, foram realizados testes variando seus parâmetros. Nos testes onde seu utilizou a técnica de comparação de histograma para escolher os melhores candidatos, variou-se as métricas de comparação com base nos resultados prévios entre as interseção e o Chi quadrado alternativo. Além procurou variar as barras dos histogramas, fornecendo mais ou menos informações para a comparação. Por sua vez com o uso da escolha do melhor candidato a placa com base na abordagem número 3, contagem do número de elementos internos a placa, buscou-se variar a quantidade de placas aceitas por meio dos parâmetros aproximação poligonal e do Aspect Ratio e o tamanho dos elementos internos selecionados para a contagem. Os resultados para essa abordagem podem ser visualizados na Tabela 3.

Tabela 3 – Resultados da abordagem de detecção 1.

SELETOR DE PLACA	PARÂMETROS	IOU MAIOR QUE 50%	ERRADO	RUIM	BOM	EXCELENTE	SOMA DE BONS COM EXCELENTES
Histograma	CMP=INTER;nbins=64	60,25	39,3	28	32,65	0,05	32,7
Histograma	CMP=CHIALT ;nbins=64	59,6	40	28	31,95	0,05	32
Histograma	CMP=INTER;nbins=128	60,6	38,9	28,25	32,8	0,05	32,85
Histograma	CMP=CHIALT;nbins=128	54,25	45,05	25,85	29,05	0,05	29,1
Histograma	CMP=INTER;nbins=256	60,3	39,15	28,05	32,75	0,05	32,8
Histograma	CMP=CHIALT;nbins=256	50,9	48,55	23,75	27,65	0,05	27,7
Contornos	APROX=4;h>10;w<60;AR-0,3-0,45	51,25	48,4	11,25	28,85	11,5	40,35
Contornos	APROX>=4;h>10;w<60;AR-0,3-0,45	53,7	45,65	13,9	28,75	11,7	40,45
Contornos	APROX >= 4; h>5; w<40 ; AR - 0,3-0,45	54,25	44,15	16,1	28,35	10,08	38,43
Contornos	APROX >= 4; h>5; w<40 ; AR - 0,25-0,5	54,6	44,3	16,3	28,75	10,65	39,4
Contornos	R = 3; APPROX >= 3; h>5; w<40 ; AR - 0,25-0,5	51,9	47,15	14,6	28,25	10	38,25

Para a abordagem de localização número 2, matemática morfológica com limiarização por Otsu e representação por rastreamento de contornos, observou-se os resultados obtidos na abordagem anterior e foram realizados dois testes. Um para a abordagem de seleção baseada em comparação de histogramas e o outro para a seleção do melhor candidato baseada nos contornos internos. Os resultado desses testes é observado na Tabela 4.

Tabela 4 – Resultados da abordagem de detecção 2.

SELETOR DE PLACA	PARÂMETROS	IOU MAIOR QUE 50%	ERRADO	RUIM	BOM	EXCELENTE	SOMA DE BONS COM EXCELENTES
Histograma	CMP=INTER;nbins=256	29,75	69,75	2,15	12,95	15,15	28,1
Contornos		54,65	45,2	3,35	23,2	28,25	51,45

Mediante ambos resultados apresentados observou-se que a métrica de comparação de histograma de interpolação apresentou uma quantidade menor de placas erradas durantes os testes na abordagem de detecção por bordas. Além disso, observa-se que a técnica das bordas com o método do histograma para a seleção classificou uma menor quantidade de placas como erradas em relação a métricas com a seleção por contornos internos, entretanto o número de placas ditas excelentes foi bem inferior. A técnica de morfologia apresentou um errado negativo juntamente com a técnica do histograma, isso ocorre porque essa técnica gera uma maior número de falsos candidatos nos quais a técnica de histograma

falhou. Entretanto, devido a melhoria feita durante a técnica de contornos, observa-se um aumento significativo nas placas classificadas como excelente. A Tabela 5 resume os melhores resultados das duas tabelas anteriores para facilitar a comparação.

Tabela 5 – Comparação entre melhores resultados das abordagens de localização 1 e 2

ABORDAGEM	SELETOR DE PLACA	PARÂMETROS	IOU MAIOR QUE 50%	ERRADO	RUIM	BOM	EXCELENTE	SOMA DE BONS COM EXCELENTES
Bordas	Histograma	CMP=INTER;nbins=256	60,3	39,15	28,05	32,75	0,05	32,8
Bordas	Contornos	APPROX>=4;h>10;w<60;AR-0,3-0,45	53,7	45,65	13,9	28,75	11,7	40,45
Morfologia	Histograma	CMP=INTER;nbins=256	29,75	69,75	2,15	12,95	15,15	28,1
Morfologia	Contornos		54,65	45,2	3,35	23,2	28,25	51,45

5.2 Algoritmo de segmentação de caracteres

Para avaliar o algoritmo de segmentação, criou-se uma base de dados com as placas da base da UFMG. Utilizou-se essa base de placas como entrada para algoritmo de segmentação. As letras segmentadas foram salvas e para avaliar sua qualidade optou-se por uma medição indireta utilizando o próprio algoritmo de OCR desenvolvido neste trabalho. Assim, comparou-se o resultado do reconhecimento do caractere com a segmentação proposta versus a segmentação do caractere baseado nas anotações da base.

O primeiro resultado obtido foi que dos 5523 caracteres existentes o algoritmo de segmentação ofereceu 4935 respostas, isto é, aproximadamente 89,3%. Esse valor não significa que a segmentação foi correta, pois pode ter ocorrido, por exemplo, do algoritmo ter demarcado regiões da placa que não são letras.

A avaliação por meio da rede funciona como uma forma de medir a qualidade da segmentação. Em condições ideias de segmentação a rede obteve uma acurácia de 80,52%. Com o algoritmo de segmentação desenvolvido esse valor de acurácia caiu para 61,4% isso significa que dos 5523 caracteres o algoritmo segmentou corretamente aproximadamente 3030 imagens, computando por volta de 55%.

5.3 Algoritmo de OCR

Para essa tarefa considerou-se apenas o uso de RNA. Além do exemplo mostrado no capítulo anterior, que obteve 80,52% de acurácia sobre dados de teste, a rede criada foi treinada novamente. Desta fez além das 5628 imagens de treino, foram utilizada 2849 imagens para o processo de validação. As imagens de validação funcionam com uma etapa de teste dos pesos durante o treinamento. Os dados de validação não são utilizados para o treinamento, apenas atuam como termômetro para rede saber se ficou boa ou não. Entretanto, sua principal função é evitar o overfitting umas vez que ela sempre adiciona novos dados a rede durante o treinamento.Esse treinamento, resultou numa acurácia de 82% nas 5523 imagens de testes.

Esse resultados são excelentes devido a não alteração do cenário. Para avaliar o OCR treinado em um novo cenário, utilizou-se a base de dados da UFPR para realização de testes. O resultado obtido foi de 51% de acurácia.

De um modo geral os resultados foram razoáveis, porém abaixo dos encontrados na literatura que utilizam técnicas mais modernas para solucionar esse problema. Entretanto para demonstrar que a base utilizada é desafiadora executou-se o OpenALPR para placas brasileiras sobre as duas mil imagens da base da UFMG. O OpenALPR inclusive possui versões comercializáveis e possui grande confiabilidade na área. O resultado obtido para esse teste foi que o OpenALPR só conseguiu realizar algum OCR, independente se estar correto ou não, de 63% das placas dessa base.

6 Conclusão

A Visão Computacional tem se tornado cada vez mais aplicável com o passar dos anos. Hoje em dia, dispositivos móveis como celulares são capazes de utilizar sua câmeras para realizar biometria de fase, reconhecer objetos, fazer OCR de equações matemáticas. Nessa área é fundamental o entendimento do campo clássico advindo das técnicas de Processamento Digital de Imagens e das mais modernas ferramentas de Deep Learning, Machine Learning, como as redes CNN. A automação de processos e máquinas muitas vez envolvem o entendimento do ambiente ao redor que pode ser feito através da câmeras. Não deixando de citar, os carros autônomos tecnologia essa que faz uso da Visão Computacional para revolucionar os meios de transporte.

Mediante a relevância desse tema para as tecnologias vindouras, esse trabalho se propôs a ser um guia de exploração sobre diversas técnicas e abordagens de PDI e Visão Computacional. Para isso, fez uso de um estudo de caso que mistura o clássico e o moderno, o reconhecimento automático de placas veiculares (ALPR). Esse problema muito bem solucionado em cenários controlados, encontra dificuldades nos ambientes externos do mundo real. Assim em um cenário complexo buscou-se aplicar diversas técnicas de PDI e Machine Learning para solucionar cada etapa do ALPR: detecção do veículo, localização da placa, segmentação de caracteres e o reconhecimento óptico desses caracteres.

Para a detecção e segmentação de caracteres explorou-se diversos fundamentos do PDI. Pode-se citar: Detectores de bordas, filtros espaciais de suavização, filtros de aguçamento, matemática morfológica, template matching, rastreadores de contornos, aproximação poligonal, transformada de Hough, segmentação por watershed, limiarização global, algoritmo de Otsu, entre outros temas.

Apesar dos resultados obtidos não terem sido os melhores, e em alguns casos não conseguiu-se chegar a resultados claros e relevantes. Pode-se dizer que o trabalho cumpriu seu objetivo de expor as técnicas e nortear futuros trabalhos na área. Além disso, poderiam ser feitas melhorias em alguns dos algoritmos e fusões de técnicas entre eles para gerar novas propostas de solução para o problema.

Os testes realizados não foram o suficiente para cumprir toda a demanda de algoritmos produzidos. Por isso, vale ressaltar que os resultados demonstrados não podem ser generalizados. Eles dependem do problema que se quer solucionar sendo assim em quanto algumas técnicas produziram bons resultados para esse cenário, outras técnicas podem funcionar melhor em outra cenário.

Ao todo foram produzidos mais de dez Jupyter Notebooks, cada um com abordagens e técnicas diferentes que podem auxiliar em uma futura disciplina de PDI. Como já

mencionado, eles abordam de forma aplicada boa parte da teoria do Processamento Digital de Sinais.

Com tudo isso, é notório, que o trabalho conseguiu expor, com sucesso ou sem, um grande número de técnicas do PDI e um inicio na programação de RNA com o TensorFlow Keras. Além disso, explorou-se um pouco o universo de Deep Learning para a detecção de objetos, citando redes como a SSD e o Yolov3 e mostrando como importa-las para uso pessoal.

Por fim, esse trabalho possui muito espaço para evoluir. Pode se aperfeiçoar as técnicas de PDI que não deram certo e até mesmo utilizar outras. Também pode-se explorar novas bases e realizar novos testes para a obtenção de melhores resultados. Contudo, ele conseguiu satisfazer os requisitos propostos em seus objetivos.

Referências

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>. Citado na página 38.

ANAGNOSTOPOULOS, C.-N. E. License plate recognition: A brief tutorial. *IEEE Intelligent transportation systems magazine*, IEEE, v. 6, n. 1, p. 59–67, 2014. Citado na página 45.

ANDREAS, K.; ABIDI, M. *Digital Color Image Processing*. [S.l.]: Wiley, 2008. Citado na página 20.

BALLARD, D. H.; BROWN, C. M. *Computer Vision*. Estados Unidos: Prentice Hall, 1982. Citado 2 vezes nas páginas 15 e 16.

BISHOP, C. et al. *Neural Networks for Pattern Recognition*. [S.l.]: Clarendon Press, 1995. (Advanced Texts in Econometrics). ISBN 9780198538646. Citado na página 34.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Citado na página 37.

BULAN, O. et al. Segmentation- and annotation-free license plate recognition with deep localization and failure identification. *IEEE Transactions on Intelligent Transportation Systems*, v. 18, n. 9, p. 2351–2363, Sept 2017. ISSN 1524-9050. Citado na página 41.

CANNY, J. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, p. 679–698, 1986. Citado na página 27.

CARDON, A.; MULLER, D. N. *Introdução Às Redes Neurais Artificiais*. Rio Grande do Sul, Porto Alegre: [s.n.], 1994. Citado 2 vezes nas páginas 33 e 34.

CHIPANA, F. E. A.; IANO, Y. Segmentação de imagens: abordagens para reconhecimento de placas de veículos. 2010. Citado na página 24.

CHOLLET, F. et al. *Keras*. 2015. <https://keras.io>. Citado na página 38.

CHUN, W. *Core Python Programming*. [S.l.]: Pearson Education, 2006. (Core Series). ISBN 9780137061594. Citado na página 36.

CIREŞAN, D. C. et al. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011. Citado na página 16.

CORNETO, G. L. et al. A new method for automatic vehicle license plate detection. *IEEE Latin America Transactions*, v. 15, n. 1, p. 75–80, Jan 2017. ISSN 1548-0992. Citado na página 41.

CULJAK, I. et al. A brief introduction to opencv. In: *2012 Proceedings of the 35th International Convention MIPRO*. [S.l.: s.n.], 2012. p. 1725–1730. Citado na página 37.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. [S.l.], 2009. p. 248–255. Citado na página 16.

DODGE, Y.; INSTITUTE, I. S. *The Oxford Dictionary of Statistical Terms*. [S.l.]: Oxford University Press, 2006. ISBN 9780199206131. Citado na página 21.

DOWNEY, A. *Think Python*. [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página 36.

DUDA, R. O.; HART, P. E. *Use of the Hough transformation to detect lines and curves in pictures*. [S.l.], 1971. Citado 2 vezes nas páginas 27 e 28.

DUDA, R. O.; HART, P. E. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, ACM, v. 15, n. 1, p. 11–15, 1972. Citado na página 20.

FACON, J. A morfologia matemática e suas aplicações em processamento de imagens. In: *VII Workshop de Visão Computacional–WVC*. [S.l.: s.n.], 2011. v. 13. Citado na página 29.

FERNEDA, E. Neural networks and its application in information retrieval systems. *Ciência da Informação*, SciELO Brasil, v. 35, n. 1, p. 25–30, 2006. Citado na página 31.

GONÇALVES, G. R. *License Plate Recognition based on temporal redundancy*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Minas Gerais, 8 2016. Citado na página 41.

GONÇALVES, G. R. et al. Benchmark for license plate character segmentation. *Journal of Electronic Imaging*, v. 25, n. 5, p. 1–5, 2016. Disponível em: <http://www.ssig.dcc.ufmg.br/wp-content/uploads/2016/11/JEI-2016-Benchmark.pdf>. Citado 2 vezes nas páginas 39 e 40.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X. Citado 7 vezes nas páginas 20, 22, 23, 24, 26, 29 e 30.

GOYAL, A.; BIJALWAN, A.; CHOWDHURY, M. K. A comprehensive review of image smoothing techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, v. 1, n. 4, p. pp–315, 2012. Citado na página 23.

HALTERMAN, R. L. Learning to program with python. 2011. Citado na página 36.

HART, P. E. How the hough transform was invented [dsp history]. *IEEE Signal Processing Magazine*, IEEE, v. 26, n. 6, 2009. Citado 2 vezes nas páginas 27 e 28.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. [S.l.]: Artmed, 2007. ISBN 9788577800865. Citado 2 vezes nas páginas 31 e 32.

HOUGH, P. V. *Method and means for recognizing complex patterns*. [S.l.]: Google Patents, 1962. US Patent 3,069,654. Citado na página 27.

JUPYTER STEERING COUNCIL. *Project Jupyter*. 2018. Disponível em: <http://jupyter.org/about>. Citado 2 vezes nas páginas 38 e 39.

JUPYTER TEAM. *The Jupyter Notebook*. 2015. Disponível em: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>. Citado na página 39.

KAGGLE, INC. *Kaggle: Your Home for Data Science*. 2018. Disponível em: <https://www.kaggle.com/>. Citado na página 16.

KHAN, M. A. et al. License number plate recognition system using entropy-based features selection approach with svm. *IET Image Processing*, v. 12, n. 2, p. 200–209, 2018. ISSN 1751-9659. Citado na página 41.

KOLEN, J.; KREMER, S. *A Field Guide to Dynamical Recurrent Networks*. [S.l.]: Wiley, 2001. ISBN 9780780353695. Citado na página 16.

KOVÁCS, Z. *Redes Neurais Artificiais*. [S.l.]: LIVRARIA DA FISICA, 2002. ISBN 9788588325142. Citado na página 33.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105. Citado na página 16.

LAROCA, R. et al. A robust real-time automatic license plate recognition based on the yolo detector. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2018. p. 1–10. ISSN 2161-4407. Citado 2 vezes nas páginas 40 e 41.

LIPTON, Z. C.; BERKOWITZ, J.; ELKAN, C. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015. Citado na página 16.

MAHALAKSHMI, T.; MUTHAIAH, R.; SWAMINATHAN, P. Image processing. *Research Journal of Applied Sciences, Engineering and Technology*, 2012. Citado na página 24.

MATAS, J.; GALAMBOS, C.; KITTLER, J. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, Elsevier, v. 78, n. 1, p. 119–137, 2000. Citado na página 55.

NIXON, M. S.; AGUADO, A. S. *Feature extraction & image processing for computer vision*. [S.l.]: Academic Press, 2012. Citado na página 20.

OPENALPR TECHNOLOGY, INC. *OpenALPR Documentation*. 2017. Disponível em: <OpenALPRTechnology,Inc>. Citado na página 39.

OPENCV. *Morphological Transformations*. 2018. Disponível em: <https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>. Acesso em: 24 junho 2018. Citado na página 30.

OTSU, N. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, IEEE, v. 9, n. 1, p. 62–66, 1979. Citado na página 25.

PEDRINI, H.; SCHWARTZ, W. R. *Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações*. São Paulo: THOMSON Learning, 2008. Citado 7 vezes nas páginas 19, 21, 22, 23, 24, 29 e 30.

PERELMUTER, G. et al. Reconhecimento de imagens bidimensionais utilizando redes neurais artificiais. *Anais do VIII SIBGRAPI*, p. 197–203, 1995. Citado na página 20.

PÉREZ, F.; GRANGER, B. E. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, IEEE Computer Society, v. 9, n. 3, p. 21–29, maio 2007. ISSN 1521-9615. Disponível em: <https://ipython.org>. Citado na página 39.

RASCHKA, S. *Python Machine Learning*. [S.l.]: Packt Publishing, 2015. ISBN 9781783555147. Citado na página 36.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. Citado na página 38.

ROSENFELD, A. Picture processing by computer. *ACM Computing Surveys (CSUR)*, ACM, v. 1, n. 3, 1969. Citado na página 27.

ROSSUM, G. van; BOER, J. de. *Linking a Stub Generator (AIL) to a Prototyping Language (Python)*. Tromso, Norway: [s.n.], 1991. Citado na página 36.

RUSS, J. C. *The Image Processing Handbook*. 2. ed. [S.l.]: CRC Press, 1995. Citado 2 vezes nas páginas 20 e 21.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence - A Modern Approach*. New Jersey: Pearson Education, 2010. Citado na página 20.

SALDANHA, M. F.; FREITAS, C. Segmentação de imagens digitais: Uma revisão. *Divisão de Processamento de Imagens-Instituto Nacional de Pesquisas Espaciais (INPE), São Paulo*, 2009. Citado na página 25.

SMITH, A. R. *A pixel is not a little square, a pixel is not a little square, a pixel is not a little square! (And a voxel is not a little cube)*. 1995. Memorando técnico da Microsoft. Citado na página 20.

SUZUKI, S. et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, Academic Press, v. 30, n. 1, p. 32–46, 1985. Citado 2 vezes nas páginas 31 e 48.

WATT, A. H.; POLICARPO, F. *The Computer Image*. [S.l.]: Addison-Wesley, 1998. (ACM Press books, v. 1). ISBN 9780201422986. Citado na página 23.